A NETWORK-CENTRIC DESIGN FOR RELATIONSHIP-BASED RIGHTS MANAGEMENT

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

R. Martin Röscheisen
December 1997

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Terry Winograd (Principal Advisor)
Computer Science Department
Stanford University

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Gio Wiederhold
Computer Science Department
Stanford University

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Carey Heckman
Stanford Law School

Approved for the University Committee on Graduate Studies:

Note: This document is a reformatted version of the thesis submitted to Stanford.

Martin Roscheisen
rmr@cs.stanford.edu

Gates Building 396
Stanford University
Stanford, CA 94305

A NETWORK-CENTRIC DESIGN FOR RELATIONSHIP-BASED RIGHTS MANAGEMENT

Martin Röscheisen
Computer Science Department
Stanford University

Networked environments such as the Internet provide a new platform for communication and information access. In this thesis, we address the question of how to articulate and enforce boundaries of control on top of this platform, while enabling collaboration and sharing in a peer-to-peer environment.

We develop the concepts and technologies for a new Internet service layer, called FIRM, that enables structured rights/relationship management. Using a prototype implementation, RManage, we show how FIRM makes it possible to unify rights/relationship management from a user-centered perspective and to support full end-to-end integration of shared control state in network services and users' client applications.

We present a network-centric architecture for managing control information, which generalizes previous, client/server-based models to a peer-to-peer environment. Principles and concepts from contract law are used to identify a generic way of representing the shared structure of different kinds of relationships.

ACKNOWLEDGEMENTS

Ten years ago, Martin Kay agreed to invite me to spend a couple of months with him at Xerox PARC, in the natural language processing group led at the time by Kris Halvorsen. I consider the completion of this thesis in many regards as an outgrowth of this time that I spent at PARC in 1987/88 right after coming out of high school, as well as during subsequent summers as part of their excellent summer internship program.

Once at Stanford, my Ph.D. advisor Terry Winograd enabled me to follow my interest in exploring the possibilities of the new Internet/Web medium that was just emerging. In the first two years, this led to two different systems for Web-based collaboration: a system for sharing annotations on arbitrary Web pages, called ComMentor (with Christian Mogensen; http://pcd.stanford.edu/ComMentor) and a system for collection-based collaboration, called Grassroots (with Kenichi Kamiya; http://pcd.stanford.edu/Grassroots).

When the time came to choose a subject for my Ph.D. thesis, Terry suggested that I reexamine the classical topic of access control. Terry anticipated that this issue would emerge as one of the key questions to be addressed in order to allow the new medium to be able to live up to its true potential. We quickly identified that issues such as privacy and access control are more about relationships than about information, and that we need to think about mechanisms to support relationship management. Now that I have completed more than two years of research and prototyping about how to make such an approach work, it turns out that others agree with the basic assumptions that have guided this work. Kevin Kelly is quoted in a recent *Time* cover story as saying that "We think that privacy is about information, but it's not—it's about relationships"; Esther Dyson devotes an entire issue of *Release 1.0* to the question of how one can deal with access control and privacy in the online domain; etc. I wish Terry would again give me two year's worth of lead time, equivalent to about a decade in Internet time—and this time I probably would not use it to get a second Ph.D. Terry also helped shape this thesis significantly through weekly meetings and through his innovative teaching program in user-centered software design.

I am further indebted to Hector Garcia-Molina for coalescing a working group around the Stanford Digital Libraries Project, and to Andreas Paepcke, Scott Hassan and Alan Steremberg for developing a testbed infrastructure that greatly facilitated my prototyping work.

My fellow Ph.D. students Michelle Baldonado, Steve Cousins, Frankie James, Luis Gravano, Steven Ketchpel and Larry Page, as well as the other members of the Stanford Digital Libraries Project, provided continual feedback from the early concept stage to the final presentation of this thesis.

As members of my reading committee, Gio Wiederhold and Carey Heckman provided numerous comments about the thesis; this improved the final presentation significantly.

Tim Stanley has helped me explore the legal and economic issues related to my thesis. A side effect of our discussions is FindLaw (http://www.findlaw.com).

In the context of two seminars, Paul Goldstein helped me crystallize my thinking about the relation between property systems (such as copyright) and contracts, which encouraged me ultimately to focus on the relational approach to rights management that characterizes this thesis.

Over the past years, I have received valuable comments contributing to my thesis (given specific parts in early versions only) from a number of individuals outside of Stanford, including Mark Ackerman, Michel Bilello, Victoria Bellotti, Lorrie Cranor, Jim Davis, Henry Gladney, Ben Gross, Johannes Klein, Ulrich Kohl, Carl Lagoze, Susan Owicki, Paul Resnick, Jerry Saltzer, Pamela Samuelson, Dave Solo, Mark Stefik, Hal Varian, Dan Wallach, Michael Wellman, Mary-Ellen Zurko.

Last but not least, I am grateful to my parents Margret and Fritz for providing a unique formative environment and support, and to Andrea—not only for being the first environmental geochemist who asked a question in a computer science thesis defense that led to significant redesign and rewriting (Chapter 4).

Martin Röscheisen
Palo Alto, November 1997

Table of Contents

List of Illustrations

1.0 Towards Frictionless Digital Rights/Relationship Management

1.1 The Problem: Rights/Relationship Management in Networked Environments

Over the past few years, the Internet has evolved from an experimental data network used by a fairly narrow community of researchers to a general medium with a good potential for mainstream usage. Every moment now, more of the basic building blocks—from network routers and connections to the different software layers and services—are being put into place, creating an infrastructure that increasingly gives reality to metaphors such as “information at your fingertips” and “contact anyone anywhere.”

The prospect of having a platform where anyone is able to get information anywhere, anytime, creates the necessity to address the fundamental question of how to articulate and enforce boundaries of control on top of this platform, while enabling collaboration and sharing in a peer-to-peer environment. We call this question one of “rights/relationship management.”

Note that the efficiencies of the online medium do not change the facts that people have specific privacy preferences, that companies need to control access to their assets, that authors might want to get compensated or at least know who is in their audience, etc. However, today’s Internet infrastructure supports such boundaries of control only in a very limited way.

As an example, consider that currently every registration-based Web service puts up its own user interface (in the form of a set of Web pages) to allow people to register and to use their registration to access certain content areas. All these interfaces try to achieve basically one and the same purpose, but they are all different and—from a user-centered perspective—offer heterogeneity that contributes to a less satisfactory overall user experience. There is also an issue of a lack of end-to-end integration. For instance, while the provider’s membership database will be well integrated with the registration function, users have no corresponding “client”-side integration that helps them manage their relationships with the providers of the various network services.

As another example, consider that what a Web server currently knows about the person behind a browser is limited to a few general attributes, such as the browser’s IP address, the type of software used, etc.; but it is not clear whether the person is an employee (who might be authorized to access company documents), a student (who might qualify for a student discount), a US citizen (who might be subject to certain export controls), a subscriber with a subscription contract, or someone else with different kinds of relationships.

This thesis is about a concrete design that defines the kind of technology that can help us address this issue of rights/relationship management in heterogeneous, networked environments. It suggests how the current infrastructure can be augmented by another software service layer—one for digital rights management—that allows us to talk about such high-level objects as “contract,” “obligation,” “right,” and “person”—and to deal with the question of how to provide better end-to-end integration for relationship management applications and how to lower the usability threshold for access-controlled services.

1.2 Current Solutions: Idiosyncratic, Not User-Centered

There already exist several mechanisms implementing specific forms of digital rights management, including mechanisms to control access to files and network services, mechanisms to limit the use of information, mechanisms to select anonymity or pseudonymity, etc. For a survey of a recent set of commercial rights-management solutions, see [96] and [98].

Conceptually, we can think of these mechanisms as falling into three classes, depending on whether they predominantly manage control information in a "server-based," "client-based," or "third-party based" way:

- *Server-based control:* This is the traditional model, widely employed for access control in file systems, Web servers, security firewalls, etc. The "access" of information is protected by having a server check the control information that it manages along with the services/information that it provides.
- *Client-based control:* Client-based control has been in use for many years. For example, "demo copies" of commercial software often have usage limitations (such as an expiration date or limited functionality) built into the code, which can then be freely distributed. More recently, the work on trusted clients by Stefik [97] has produced a far more general version of such client-based control for use in consumer content commerce.
- *Third-party based control:* The Copyright Clearance Center is an example of control that is managed by third parties. Other examples include the "license servers" that are now routine purchase options for commercial software such as FrameMaker, PhotoShop, etc.

The overall picture is that we have a disparate set of special-purpose solutions, often implemented in a proprietary way. In networked environments, such heterogeneity easily translates into interoperability problems and a lack of symmetric end-to-end integration. Furthermore, by virtue of the fact that each system has a different control interface, we do not provide users with an interaction model that is uniform from their perspective. Finally, the lack of a common platform also makes it more difficult, if not impossible in certain cases, for application developers to introduce new kinds of control behaviors for the services that they want to make available. The following example illustrates some of these points in more detail.

1.3 Unifying Rights Management in a User-Centered Way: Example

Consider the publisher of an online newsletter who puts a system in place to manage subscribers and give them preferential access to certain content areas. The user interface for such a system would typically consist of a set of Web pages designed by the service provider. In such an interface, users find a way to fill out HTML forms with fields for their name, address and payment information, and hit some kind of a "subscribe" button, that will then trigger certain actions at the server side to add the person to the subscriber database, schedule invoices, etc. In other words, the system is designed in a way that integrates well with the service provider's backend infrastructure, such as membership databases and payment processing applications. From the provider's perspective, this is an obvious approach to take.

However, from a user-centered perspective, we are exposed to lots of different interfaces for the various services that all try to achieve basically the same purpose. The current situation on the Web is essentially like a corresponding (theoretical) situation for personal computer applications in which every developer of a PC application designs their own way of scrolling a window and of copying and pasting text segments. This can easily lead to confusion. Note that even if two Web registration interfaces look and behave similarly, we cannot be sure whether their underlying terms and conditions are similar. For instance, two identical interfaces for different services can easily have widely differing privacy policies regarding the use of user-provided personal information. There is in general no structured way by which one can get hold of such additional properties.

A second set of issues relates to the asymmetric way in which integration is provided with each of the participants' systems. Interestingly, while a provider's membership database will be well integrated with the subscription function, there is almost no support available for having corresponding integration available on the user's side, which is also called the "client" side. Given that a contract/relationship is fundamentally a symmetric arrangement, one would expect that there ought to be about an equal number of useful actions that can be triggered on the client side. For instance, it would be useful for subscribers to list their current subscriptions in a contract portfolio, to allow them to schedule how to live up to a payment obligation from this listing, to provide an easy way to examine the state of a relationship, to terminate it, etc.

In fact, the program by which users participate is called a "client" application—a term that does not necessarily suggest a design with the kinds of peer-to-peer symmetries that characterizes many Web-based interactions today. We still call them "client applications" because client-server has been the traditional metaphor and because the current infrastructure is limited in a way that does not allow full end-to-end integration that would give these client applications a richer and more structured set of affordances. For example, consider that while a publisher can easily manage a subscription relationship by accessing a membership database directly, users would probably have to use a toll-free phone number for requests to cancel the subscription; for invoicing purposes, the postal system probably would have to be used; etc.

To summarize, a currently fairly typical solution to Web services is characterized by

- *low internal integration:* There are disparate interaction channels to manage one and the same relationship;
- *low external integration:* An authorized outside party (e.g., the subscriber with a client application) has no structured way to access information about the state of a relationship other than through a (non-programmatic) Web interface or through one of the Internet-external channels; and
- *user interface heterogeneity:* Every content site that offers subscriptions has its own interface to accomplish one and the same type of transaction. In many cases, such heterogeneity will signify inconsistency. For instance, at some sites, clicking on "subscribe" signifies the actual acceptance of a legal agreement, while at others this only leads to another dialogue that then describes the actual offer.

In this thesis, we propose a solution to these issues that makes user interfaces uniform from a user-centered perspective and that enables full end-to-end integration. If we had

to deal only with a small number of user interface providers, we could use a proxy-based solution with a proxy for each of these providers, which would then map the idiosyncratic interaction models into a more uniform framework. However, given the fact that there is a very large number of interface developers, this is not a realistic approach. The approach to take then will have to be based on augmenting the Internet infrastructure with an appropriate network service layer and by making available toolkits that make it easy for application developers to follow certain guidelines.

In other words, by having an infrastructure that "opens up" proprietary rights-management solutions, we obtain a structured way of talking about control information, such as access rights and contracts, across different applications and services. This suggests a shared format for rights-management interactions that can propel better user interfaces in the same way that the Macintosh Human Interface Guidelines [243] and the associated toolbox that came with the Macintosh OS were able to enforce interface standards across applications from different developers, and made the Macintosh easy to use for less sophisticated computer users. For instance, by introducing a development platform that made it easy for developers to standardize the look and feel of scroll bars in their applications, the Macintosh was able to provide a better user experience than other systems.

A toolkit in conjunction with the APIs/protocols of the FIRM service layer proposed and prototyped in this thesis would have a number of benefits: Application developers could rapidly implement the desired control behavior by using components provided by the rights-management toolkit. Moreover, the resulting solution will have higher degrees of internal and external integration. Since the rights management service layer provides for standard types of affordances with respect to contracts (accepting, terminating, etc.), it is easier to provide a more comprehensive interface that also deals with cases that are otherwise often left out, such as means for cancelling a contract, returning goods, etc.

Furthermore, interested third parties can make use of the service layer to interface with the appropriate rights-management structures directly. For example, someone could easily develop a next-generation Quicken-type application that provides affordances for managing contracts and the rights and obligations that result from them. Such an application could provide an interface for cancelling contracts that one signed up with different providers. It could also directly deal with electronic invoices from different contracts. Ultimately, this enables us to shift the perspective from provider-centered user interactions, where users interact with Web forms put up by every publisher, etc., to a user-centered view, where users interact with a relationship manager application that has direct access to the relationship held with a publisher. The RManage system that we shall describe in this thesis is a prototype "relationship manager" application based on our FIRM rights management service layer, that gives a first glimpse of such an interface.

1.4 Defining a Rights Management Service Layer

Architectually, we achieve the kind of unification of services and protocols necessary for networked rights management by defining a network software service layer that is built on top of other network protocols to provide object definitions and services for managing rights and obligations.

Three Classes of Usages

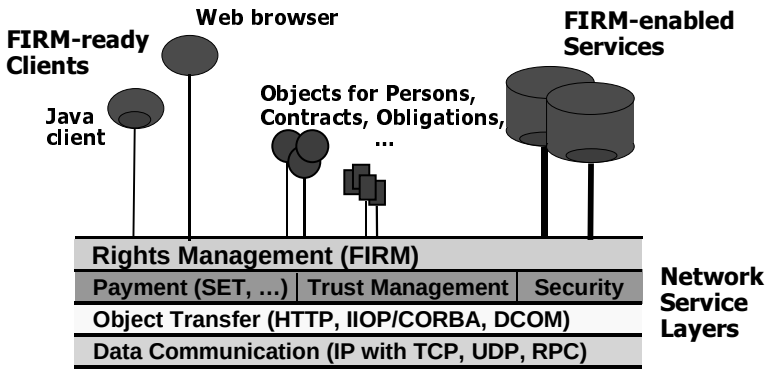
Objects provided by a rights management service layer essentially have three major classes of usage:

- *Authorizing/Controlling Actions:* There is a way to determine whether a particular action is acceptable or required under a given set of rights and obligations. It is possible to execute a generic set of transactions on each object, such as fulfilling an obligation.
- *Inspection by Humans:* There is an interface through which people can obtain information about the state of a certain rights relationship, the nature of outstanding obligations, the terms and conditions for contingency cases such as return of goods, etc. Furthermore, there is a clear mapping between computational objects and their interpretation within a legal context.
- *Use by "Agents":* Certain client programs ("agents") can interface with rights-management objects in a structured way. For example, a program might search for the "best" offer for something by interfacing with the state information that specific rights objects make available, such as terms and conditions about pricing, return of goods, etc.

Our Solution: The FIRM Rights Management Service Layer

The rights management service layer proposed in this thesis is called the Stanford Framework for Interoperable Rights Management (FIRM). FIRM is one of the protocols of the Stanford "Infobus," a prototype infrastructure developed as part of the Stanford Digital Libraries Project that is designed to provide a way of extending the Internet protocols to higher-level information management protocol. For a survey of the architecture and the five service layers of the Stanford Infobus, cf. [253].

FIGURE 1. FIRM defines a rights management service layer on top of other Internet protocols.



FIRM makes use of a number of other services such as a standard attribute service [242] as well as services provided by the Stanford Infobus for managing items and collections, and for managing metadata. The FIRM architecture, however, is general and in no way dependent on the specific protocols implemented as part of the Stanford Infobus; it is also possible to use FIRM as an add-on to conventional Web servers, e.g., to use the

W3C's P3P privacy vocabulary [219] instead of the one used in the Stanford prototype, etc.

Related Service Layers: Security and Trust Management

Rights management is a higher-level service layer that builds on the availability of a set of other network layers. In addition to the basic network layers in the capacity of the Internet Protocol (IP), the basic Web object transfer protocol (HTTP), and various distributed object protocols (IIOP/CORBA, DCOM), this includes layers that provide services for security and digital trust management:

- *Security* is concerned with the problem of assuring the authenticity and integrity of information. Fortunately, an array of protocols are now widely available, such as security service layers from Intel [223], Microsoft [224] as well as security standards and security toolkits, such as SSL [228], S-HTTP [229] and the Java security library [230]. At the security level, we can therefore use best-of-a-kind solutions "out of the box" for our purposes.
- *Trust management* addresses the question of how to represent people's trust preferences about issues such as whether a piece of code (an "applet") is trusted to execute on a certain machine. Examples include AuthentiCode [231], signed applets [232]; PolicyMaker [135], Referee [142].

Our approach to rights management keeps both security and trust management, as well as a whole set of more basic services, orthogonal to the services of a rights management layer. This keeps the rights management service layer "thin" and simple. Of course, specific implementations will have to make use of security services, and they will also have to incorporate various forms of trust management—both at the service implementation level and at the user interface level.

For example, as we shall see later, our digital rights management service layer provides for digital contract objects that are interpreted at specialized "control object manager" servers. Security mechanisms ensure that only authorized people can modify the state of a specific digital contract. Furthermore, trust management would be used to allow people to decide whether a specific server is trusted to manage a certain digital contract.

1.5 Overall Design Assumptions

In the design of FIRM, we have taken into account several basic properties of current networked environments such as the Internet. These include heterogeneity of trust, multiplicity of enforcement choices, and multiplicity of implementation mechanisms.

Heterogeneity of Trust

Differences in levels of trust in different areas of the computing infrastructure are prevalent in networked environments such as the Internet. Even machines on the same Local Area Network within an organization carry different trust levels. These differences might be small, but they are still significant enough to lead to different practices and architectures. For example, to enforce licensing requirements it would be conceivable (and advantageous) to manage individual licenses for commercially licensed software directly at the machine where the software is used. However, the predominantly established practice is to have a license server run on another machine: a server that is often

on the same local network, but that is administratively not as easily accessible to users as their own workstations.

We assume that such differences in levels of trust will continue to persist in networked environments. For general use, we cannot expect to have one homogeneous trusted computing base. [This assumption may well be inappropriate for specific other domains such as a new product line of video players from Sony, a network of copier services by Xerox, etc.]

Multiplicity of Enforcement Choices

Mechanisms for enforcing boundaries of control include, but are not limited to, enforcement by technical locks, enforcement by police, prevention, fail-safe design, monitoring, reputation-based and "panoptic" control (see page 18).

Often it is more effective to use enforcement means other than technical locks. This is especially the case in contexts where social mechanisms are already in place that can be leveraged in enforcement. For example, software piracy is illegal independent of whether or not the software is specially protected, and the threat of audits in combination with appropriate systems of policing has minimized software piracy in businesses in the United States quite effectively. Monitoring is another type of enforcement that addresses some concerns quite effectively. For example, rather than controlling the use of an individual document in great detail, a publisher might just want to be assured that no excessive copying is taking place.

Our assumption is that the current multitude of enforcement choices needs to be supported in a rights-management service layer. That is, we would like to provide a programmable framework for different kinds of mechanisms.

Multiplicity of Mechanisms

At a system level, we assume that designers of rights-management solutions will continue to make different design trade-offs to best deal with the specific usages at which their application is targeted. Any more general system is likely to end up being less efficient for specific uses. A part of this assumption is the fact that we have legacy systems that now need to be tied into a larger infrastructure without requiring them to be redesigned from the bottom up.

FIRM is therefore based on the assumption that the rights systems landscape in a networked environment of autonomous resources such as the Internet will continue to be heterogeneous. There is not going to be one single rights system that covers all usages and domains universally.

In other words, the assumption is that we will continue to have rights systems

- in legacy systems (e.g. the file access rights in Unix, Windows NT, etc.; the payment obligation processing in Dialog, Uncover, etc.; the PhotoShop group license server),
- from different vendors (InfoSafe, Xerox, InterTrust, ...),
- for different domains (e.g. for privacy, for parental control, etc.), and
- for use with different devices (printers, PCs, etc.) and with different media (hard disks, DVDs, etc.).

But at the same time, we would like to achieve more uniformity and a way for different kinds of applications to interoperate.

1.6 Design Process

At the heart of coming up with an appropriate design for our service layer is a process of *reification* of existing objects and usages in the cyberworld: We enrich the technical infrastructure by objects that recreate, in the cyberworld, aspects of existing objects in the way we know them. For example, we can have a digital contract object to represent the terms and conditions of a certain contractual relationship. We could design such an object in a way that it affords a "terminate" action—to reify the fact that contracts can generally be cancelled. Furthermore, we can additionally choose to reify certain practices. For instance, we can reify the practice that contracts are often based on standard templates that people only customize and fill out—leading to a design where digital contracts systematically be instantiated from contract-forms objects, a user interface where contracts can be drafted by "taking" such forms and manipulating them, and an underlying institutional infrastructure where we have "forms designers" as an independent entity.

This thesis describes an initial design for what these objects are, how they behave, and how they are grounded in the heterogeneous set of mechanisms that currently characterize the technical landscape. It is a specific design, demonstrated in a prototype infrastructure, that embeds our assumptions on which usages are of primary interest, and which ones are not as important. The main assumption along these lines, as we shall explain in the next section, is that we support relationship-based interactions.

1.7 Taking a Relationship-Based Approach

The conceptual core that we use to realize architectural unification in our FIRM service layer lies in the relationship-based perspective that we uniformly apply. Our basic approach is to shift the perspective from (information) objects to the participants' relationships.

Rather than thinking primarily in terms of a traditional "information access" model where users access information/property (and we then conceive of ways of protecting this information from being accessed in certain cases), we apply a communication model to think of the relationships that providers and consumers of information might usefully want to engage in. We then deal with issues such as privacy, security, access control, etc. as the ancillary of successfully managed relationships.

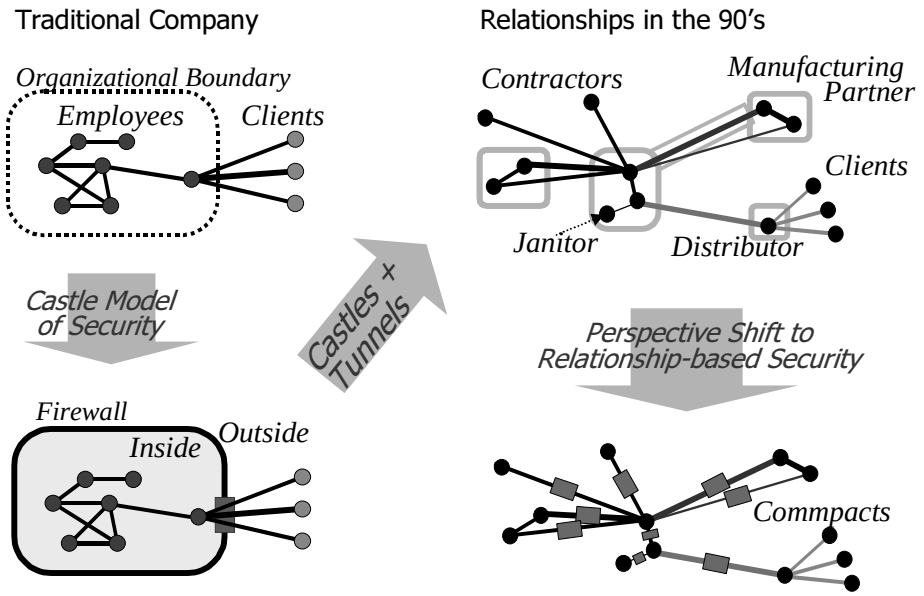
In our view, events such as privacy intrusions, security breaches, unauthorized access, etc., are primarily surface forms of an underlying process that points to unsuccessfully managed relationships between the relevant communication participants. Consequentially, our hypothesis is that by providing support for a social mechanism of coordinated expectation through relationship management, we can deal with many of these issues more effectively than if we look at these events in isolation. The following example from a specific domain illustrates our relationship-based perspective.

Example: Relationship-Based Network Security

The traditional metaphor that underlies much of network security is basically a "castle model." Security firewalls are a well-known instance that implements this model: they

construct a "wall" around an organization with access limited to certain controlled entry points that serve to protect the inside (employees, etc.) from the outside (attackers, etc.).

FIGURE 2. Network Security: FIRM enables a relationship-based approach to network security.



Even in conventional environments, this idealization creates anomalies that lead to lower levels of usability and security. For example, one of the outside persons might not be an attacker, but might in fact be the company's CEO trying to catch up with his e-mail while waiting for a connecting flight at an airport, or it might be a telecommuter, a supply-chain partner, etc. On the other hand, persons inside the firewall boundary might include individuals who have relatively weak forms of relationships with the organization, such as contractors, summer interns, etc. For these cases, the castle model only provides minimal control.

Such initial anomalies become more critical in the context of organizations that have a number of outside partners that are tightly integrated into the "inside," as with many organizations today, especially with "virtual companies," which have a smaller core company whose assets are not as much kept in a mainframe in the headquarters building, but are determined by the quality of the relationships a company has with its various partners.

Continuing to apply a castle model of security to such an environment leads on the one hand to a multitude of smaller security segments ("logical firewalls"); on the other hand, it creates the need to connect these various segments by "tunnels," such as the "extranets" that have recently received substantial attention as one way to implement such tunnels. While such an approach is certainly able to fix a good number of immediate problems, it clearly does not grasp the underlying dynamics of the situation: the fact that

genuine security needs to be relationship-based.[1] In other words, the question is how to realize collaborative extranets that are managed on a peer-to-peer basis.

By providing an infrastructure such as FIRM that radically reduces the transaction costs of managing one-to-one relationships, FIRM-based systems such as RManage effectively make it possible to implement a security architecture that allows us to shift the perspective entirely into a relationship view: Relationships (lines in Figure 2) are negotiated between communication participants (small balls in Figure 2), and encapsulated in communication pacts ("commpacts") whose externalization can in principle then reside anywhere on the network (small boxes in Figure 2). Security is then obtained as the ancillary of successfully managed relationships, rather than by imposing property-based boundaries.

Other Examples

Relationship-based security is by no means limited to the problems that motivate the use of firewalls. Much of what falls into the domain of contracting is subject to relationship-based interactions. If a subdivision of a company wants to establish a new relationship with an external supplier that includes sharing a significant amount of information (indeed possibly even more than the same subdivision shares on a need-to-know basis with any other group in its own organization), then we have a relationship issue that cuts across organizational boundaries.

We need to support the kind of "trusted sharability" that has been described as being the most significant impediment currently limiting the adoption of otherwise preferred organizational forms (cf. [46] on virtual companies). Similar issues come up in privacy-related areas, such as making health records available in a controlled way as part of the highly structured relationships that now characterize the health-care industry after its transformation from a largely invoice-based, service-for-fee cottage industry to one based on prenegotiated contracts with Health Maintenance Organizations and Integrated Delivery Systems [91]. Flexible records management requires managing dynamic but structured relationships: that is, relationships that are not static to predefined organizational boundaries, but that also have more structure and context than have been considered in current models for simple exchange transactions. (See [29] for a perspective on how to understand corresponding "relationship marketing.")

Trusted sharability can have quite light-weight appearances. For example, consider finding information about the current weather. Curiously, the interactive experience that one currently gets on the Web is in some regard worse than what one gets by just tuning into conventional broadcast TV in that, in the TV case, one at least obtains directly the weather one cares about—without first having to go through a number of steps by which one narrows down one's geographical location. Having a structured, controlled way by which servers can automatically obtain a user's ZIP code for defined purposes would allow the underlying "one-to-one infrastructure" to actually cash in on its promise and provide actual "one-to-one content." The same holds of course for many other examples, including having a consensual and controlled way of allowing publishers of adver-

1. Note that the word "relationship" is used in many different ways in different contexts ("customer relationships", etc.). In this thesis, we use the expression in a fairly general sense, designating a pattern of interaction between communication participants that is based on structured, mutual expectations. For instance, a tacit understanding between two persons would qualify as a relationship in our usage here.

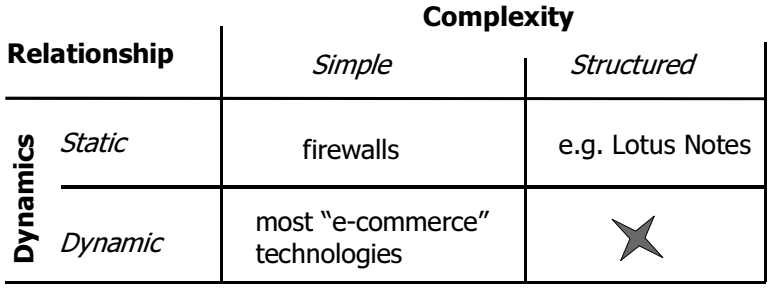
tising-based Web sites to collect certain basic forms of demographic information for their advertisers, etc. (See the P3P work for further usages [219].)

1.8 Design Space

One way to understand the design space is to consider relationships along two dimensions:

- their *dynamics*, or how relationships are formed and completed,
- their *complexity*, or how much information one would need to characterize a relationship.

FIGURE 3. Design Space: Systems designed for different types of relationships.



Different technologies address different corners of this space. In particular, security firewalls are both simple (one is either "in" or "out") and fairly static (their boundaries only change as a result of reorganizations or acquisitions). Most "e-commerce" technologies currently address mostly fairly simple—although possibly quite dynamic—relationships. For example, purchasing an online image by using a micropayment mechanism is a relatively simple exchange transaction, but there might be many of them in a relatively short time period. Micropayment mechanisms can of course be used for fulfillment tasks that come up in a relational context such as subscriptions, but the mechanism itself does not support this context.

At the other end of the complexity spectrum, we have systems such as Lotus Notes that deal with more structured relationships. After all, the target customers of such systems have been enterprises that want to make their internal processes more efficient, as a first step before addressing the question of how to improve their interactions with outside parties.

Of course, both systems that were initially designed for usages in the static/structured and in the dynamic/simple corner tend to have efforts on the way to extend their capabilities to the dynamic/structured corner—the position in the design space with which we are primarily concerned. Also, while trying to accommodate both structured and dynamic relationships, we would clearly like to arrive at a design that also supports less dynamic and less structured relationships just as effectively with the same mechanisms.

1.9 Design Goals

The goals of our design are therefore to address the following questions:

- What is the overall conceptual framework that gives us a structured way to think about relationships and how they relate to performing actions?
- What is a corresponding system architecture for managing control information that we can use to apply this framework in the context of a networked environment such as the Internet?
- At a systems level, how do we deal with domain extensibility and interoperability given the wide range of possible relationships and mechanisms? Specifically, what is the specification for a corresponding rights management service layer?
- Finally, what components are needed to make the system easy to use?

1.10 The Solution: Outline and Summary

Let us summarize key aspects of the design that we propose in this thesis as a solution to the above goals.

A Conceptual Framework for Relationship Management: Commpacts

We articulate a communication model that situates actions in the context of previously negotiated relationships. Relationships serve as social reference points that constitute a baseline with respect to which communication participants perform and evaluate actions. In our model, every action is therefore performed with respect to a "communication pact" (short: "commpact") that encapsulates the boundary conditions of a social relationship. Possible externalizations of commpacts include but are not limited to legal contracts and informal conventions. We also outline how this model offers a relationship-based generalization of speech act theory.

An Architecture for Managing Control Information: Network-Centric

We develop an architecture for managing first-class control objects (commpacts) in a way that reflects the basic notions of our conceptual framework. This architecture allows commpacts to reside in principle anywhere on the network (which is why we call it a "network-centric" architecture), allowing them to provide control in principle to any other network object. This accommodates a wide range of usage scenarios, and generalizes previous, client/server-centered models of access control to a peer-to-peer environment.

A Structured Way of Representing Relationships: Reifying of Contract Law

At the next level of detail, we examine the question of exactly how rights relationships are structured and articulated. We rely heavily on contract law as the body of principles and concepts that describe the shared structure of different rights relationships. We describe the objects and protocols that define the core of our service layer. By separating generic from domain-specific elements, we ensure that the architecture is extensible to arbitrary domains and rights relationships.

A Demonstration Prototype: The RManage Relationship Manager

Finally, we describe a relationship manager application, called RManage, that is enabled by the FIRM service layer. RManage unifies rights/relationship management from a

user-centered perspective, and it supports full end-to-end integration of shared control state in network services and users' client applications.

RManage augments services such as Web servers with a component that allows these services to make use of the FIRM infrastructure. RManage also provides implementations of the person and contract objects that FIRM assumes. (See also Figure 1).

The RManage implementation is based on distributed objects in Java and Python, using the CORBA implementation: Xerox PARC's ILU system [246]. RManage can be used either with a plain Web browser or integrated into DLITE [249], a Java/CORBA-based direct-manipulation user interface developed as part of the Stanford Digital Libraries project. RManage implements FIRM as one of the five service layers of the Stanford Infobus [253].

RManage enables services to make available information governed by FIRM-compatible digital contracts. The sample contracts currently available include various forms of subscriptions, site licenses, and pay-per-view contracts, each with different forms of search rights, approval rights, notification obligations, and payment obligations. These digital contracts are "smart contracts" in that they integrate behavior related to what the contract is about, including authorization, payment, privacy protection, etc. For example, RManage provides fulfillment processing for a range of payment obligations by making use of the UPAI payment application interface [251]. This interface was also prototyped as part of the Stanford project; it provides an abstraction layer to integrate native payment protocols from a variety of providers such as First Virtual, DigiCash, VISA, etc.

Services that are currently using digital contracts as part of our experimental Infobus testbed include our Infobus proxies to the Dialog databases and to a document summarizer at Xerox PARC (running behind the company's firewall). Web-based services that have been augmented by FIRM plug-ins include a Web site with weather information; sample contracts deal here mainly with the controlled use of personal information.

Finally, RManage provides users with a uniform interface to the relationships that they have with the various providers of FIRM-compatible network services.

2.0 A Conceptual Model of Relationship Management

In this chapter, we articulate a framework for relationship management. We explore the notion of an agreement, abstract it into a conceptual model, and address the question of enforcement. This prepares the ground for going to the next levels in Chapters 3 and 4, where we will define a computational architecture and specify a corresponding network service layer.

2.1 Understanding Agreements/Contracts

Agreements/contracts are fundamentally a socially coordinated construct employed to frame relationships, give them structure, and set common expectations. [9][10][41] While simple exchange transactions in "spot market" environments have historically been dealt with quite effectively by simple property claims, such as in barter-based societies, contracts and contract law began to develop significantly in environments where relationships had to be clarified. They became ubiquitous in modern legal frameworks that conceptualize the existence of a contract even for simple kinds of exchange transactions.

Let us summarize here a few points that are essential for understanding what agreements are about at a general level.

- Agreements are a set of enforceable promises between two or more parties. They provide the context necessary to characterize relationships, even if they are of longer-term nature; they encapsulate boundary conditions of relationships, and they create social reference points to which people can refer back at any later point, to call into presence what they had coordinated themselves about, as part of a social mechanism of coordinated expectation. See MacNeil [4][6][5] for more material along the lines of the kind of relational perspective that we assume here, and [7][8] for critical reviews that qualify this approach.
- Agreements can be enforced through a variety of means, not necessarily through the legal system only. In fact, legal enforcement is secondary in many regards to other forms of enforcement, and we do not rely on it as one of the primary mechanisms of interest here.[1] In Section 2.4, we will examine the issue of enforcement in further detail.
- Agreements provide a conceptual separation—and thus flexibility—between coordination activity (negotiation; agreeing on what to do) and fulfillment activity (performance; actually doing it). For example, in a purchase transaction, the act of coordinating oneself around the expectation that one party will pay and the other will deliver is separated from the actual fulfillment of these obligations. In other words, we have obligations as first-class objects of a language that allows us to articulate how to constrain actions and their sequencing. For instance, we can require that a payment obligation be fulfilled before an access right can be exercised, or vice versa. In fact, obligations are objects that can also be transferred, traded, etc.

1. There are at least two reasons for this. For one, the reality in the United States is that courts are basically already busy with criminal cases, leaving them virtually no room for contract matters. Secondly, there is a wide range of present and historical examples that show how contracts have been successfully used as a coordination mechanism, even in the absence of any legal enforcement. See Ellickson [47][48] and Greif [38][39][40].

- Agreements can be about objects, but they also uniformly extend to purely relational forms that are not about any objects. Agreements can easily "quantify" over multiple objects. For example, a subscription agreement can be about a whole series of items; there needs to be only one such agreement pointing to the objects about which it is. The same could be achieved in a property-based model only via extensive replication. Indeed, agreements can express constraints about objects which do not yet exist. For example, subscription agreements are usually about issues which still need to come into existence; nevertheless, we can already talk about these rights and obligations of future objects, pay for them, etc.
- Agreements provide a uniform way for adding reservations and special clauses, including warranties, guarantees, terms and conditions, etc.; this includes various forms of "strings attached" such as usage conditions. Note that the conventional subject-object model of access control created a gulf which led to the need to separate out "access" and other kinds of usage control. A relationship-based (contract) model lends itself to uniformly extend to usage control issues and to obligations and liabilities along with access rights.
- Agreements are, at least in principle, peer-to-peer, not supplicant-granter. The conceptual shift towards centering access/action control around relationships and towards a communication model instead of the requester-granter metaphor rephrases the old access-control question of "Do I grant this?" to the new questions of "Based on which relationship are we talking to each other?" and "How can we collaborate across organizational boundaries and communicate clearly to everyone what the mutual expectations are?" It recasts the access control question from that of a unilateral decision to a matter of agreeing on boundary conditions of a relationship. In other words, we recast security and access control as an issue of relationship management and collaboration.

2.2 The Commpact Model

Our framework is based on a communication model in which actors perform actions and communicate with each other about the world. Communication acts take place between actors within the context of the boundary conditions of the previously negotiated social relationship between them. These boundary conditions do not necessarily have to have been explicitly articulated anywhere; they could be just in the form of a tacit understanding, for example. However, we postulate that there always exists an agreement on the boundary conditions of the communication relationship—thus only making it possible for effective communication to take place—and if this agreement is imperfect or if it could be more specific, then there will be negotiation communication to the effect of finding a better agreement.

At the risk of providing a bad metaphor, we can compare this situation to that of a radio receiver that is being tuned into the right frequency to be able to receive signals from a sender—only that we have a more peer-to-peer situation here where both the sender and the receiver have a part in adjusting their behavior in order to find a common ground for communication, in a way that is negotiated in the same medium in which all of the rest of the communication takes place as well.

There are two distinct processes of communication going on: a.) negotiation of the boundary conditions of a social relationship, and b.) performance within the context of

the agreed-upon boundary conditions of such a relationship. As a matter of terminology, we shall refer to this set of conditions that frame a relationship as a "communication pact" (or "commpact" for short). We introduce this new word so that we have a way of clearly referring to the meaning in the context of our conceptual model and, later, of our computational reification. In other words, commpacts encapsulate the boundary conditions of the relationship of two or more communication participants—in a way that is able to prove a social reference point for coordination activity.

Note that commpacts can have different forms of externalizations. For example, a major subset of commpacts has an externalization in the legal world in the form of legal contracts. Other commpacts have quite different externalizations and associated enforcement means. Anonymous ftp on the Internet, for instance, is a commpact that says that people who identify themselves with their e-mail address can access the public directories of a file archive provided by the offeror of the commpact.

In the remainder of this section, we examine more closely the basic dynamics of models of negotiation and performance, which we will then further refine into a specification in Section 5 and in the Appendix:

- The negotiation model is that subjects negotiate the mutually agreed-upon boundary conditions of their relationship, and then encapsulate them in a social reference point that we call a commpact.
- The performance model is that every action is conducted with respect to a commpact chosen by the performer. This designated commpact establishes the baseline with respect to which actions are then interpreted and evaluated.

In practice, negotiation and performance often occur in parallel. Moreover, the step of designating a commpact is often largely implicit, and default rules are used to make this designation step efficient. For example, external circumstances such as the environment in which an action takes place (e.g., office vs. home) will often make clear which commpact applies by default—unless explicit other (e.g., linguistic) cues are used to introduce a different applicable commpact.

2.2.1 Negotiation Mode: Establishing Mutual Assent About a Commpact

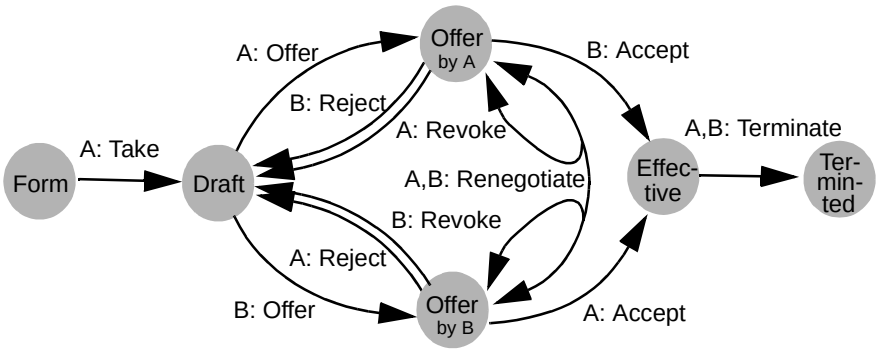
A commpact is a set of promises that is agreed upon as a result of a negotiation according to a general protocol that does not depend itself on the content or the domain of the specific promises. This protocol effectively mirrors the concepts and principles applied in contract law [1][2]. It is fundamentally based on an extension of the speech act model in Winograd & Flores [240].

The basic actions are that of issuing an offer, negotiating it, and accepting or rejecting it, or revoking it (by the party who issued it). Successfully formed, "effective" agreements can also be terminated and renegotiated, in which case a new offer takes the place of the previous offer, and a new negotiation is started. Figure 4 shows a finite-state diagram that defines the sequences in a negotiation process leading to successful contract formation.

2.2.2 Performance Mode: Making Use of an Established Commpact

The "normal" mode is that an actor would like to perform an action based on a previously established relationship. For example, a browser might want to access a search engine based on a licensing agreement that a user had set up.

FIGURE 4. Negotiation: States and Transitions.



As mentioned above, in such a case, the actor would request the execution of an action while designating the commpact with respect to which the action is supposed to be performed. This commpact would then serve as the mandate (technically: the "authorization monitor") for the given action, if the commpact is legitimate: A right in the commpact would be *exercised* to be able to perform a corresponding action. For instance, in our search engine example, in order to execute a search, we would exercise the search right of the licensing commpact.

In other words, at the performance level, we are now dealing with the issue of identifying rights that we can exercise in order to "authorize" a certain action, or identifying obligations that require certain actions to be performed. As an example, consider that a purchase agreement would generally contain a payment obligation as well as a delivery obligation and other terms and conditions. This payment obligation can be fulfilled in any of a variety of ways: by paying in cash, by sending a check in the mail, by doing an online transaction, etc. The exact, domain-dependent way in which this is done is *not* part of the rights management service layer (FIRM) that we define in the this thesis; it is left to implementation-specific mechanisms. Commpacts only talk at the level of whether or not a certain obligation has been declared to be fulfilled. FIRM provides APIs that allow us to talk about rights objects and their transactions; the way in which such transactions are performed is left to specific implementations.

As an example of the kinds of constraints that can be expressed at the rights level, consider that someone might want to require "prepayment" for a certain document access. The way to articulate this in our model would be to posit a commpact containing an access right and a payment obligation such that there exists a promissory condition for being able to exercise the access right that says that the payment obligation had been declared to be fulfilled.

2.3 Related Work

In the linguistics community (philosophy of language), speech act theory (cf. Searle [238], Austin [239]) pioneered the idea that there exists a set of generic "speech acts" that describe an expression's underlying action at a level independent of the linguistic form that realizes it. According to this framework, people communicate by issuing speech acts through which they "request," "declare," or "promise" something to someone—to name just three.

Winograd & Flores [240] took this approach further by noting that speech acts do not occur in isolation, but as part of patterns of sequences of acts that define a "conversation for action" (cf. Figure 5.1 in [240]). Like speech act theory, this model takes individual promises into the domain of first-level actions rather than separating the negotiation of a complete set of mutual promises into a distinct process. The drawback of this was that even simple purchase transactions cannot be covered in a natural way. Consider a situation where a buyer agrees with a seller to pay for a good in exchange for the seller delivering it. These are two interlinked promises: a payment obligation and a delivery obligation. However, the model in [240] does not have a mechanism to express such linkage. Another point is that the model in [240] is geared towards an organizational context, where actions always already have one default commpact, say, in the form of the employment relationship. In such contexts, it is plausible to have a model like [240], which uses issuing a 'request' as one of the possible, initial actions. The commpact model makes this explicit and provides us with a more general framework to model such actions.

The commpact model articulated in this chapter addresses this issue by providing a more powerful framework that leverages the kinds of conceptualizations that have guided work in contract law [1][2]. The idea is to conceptually separate the two distinct processes of negotiation and performance: A complete set of mutual promises is negotiated in a process that is distinct from the process by which any of the agreed-upon promises can then be fulfilled. By applying concepts and principles from the legal domain, we also make sure that actions and states are computationally reified in a way that provides a clear mapping to the way in which social behavior is considered to be structured by the legal framework.

2.4 Enforcement

In this section, we examine the question of how to enforce terms and conditions articulated in commpacts. Our main point will be that enforcement is a global property that depends on a range of dimensions that can be taken into design considerations.

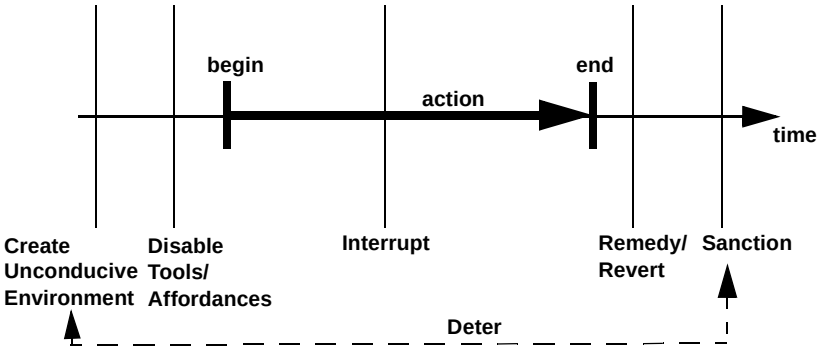
We will first lay out the types of enforcement paradigms that play a role in general. We then examine how this compares to the models of enforcement that are typically applied in the design of software applications. As a prototypical example, we describe the ISO Access Control Framework [94], which formulates an abstract control model based on the idea of interrupting "unauthorized" actions to keep them from completing. We then use a number of examples to point out how this is too constrained a model to be able to deal with a range of practical situations. This leads us then to shift to a different framework that opens the space for toolkit-based design of application-specific solutions.

2.4.1 Types of Enforcement: A Top-Down Perspective

Figure 5 gives an overview of the general types of mechanisms that can be used to enforce policies at various time stages of an action that is supposed to be controlled in some form. Table 1 then summarizes this with two examples.

Enforcement starts with the general environment. For instance, if the environment has a balanced wealth distribution and there are no enemities, then people might not even think of doing certain actions such as shooting someone or manevolently deleting someone else's files. In the "real world," we have many ways in which this type of enforcement is included, from "good neighborhoods" to open-doors offices in protected buildings, etc. In computer systems, this has not typically been part of the security thinking, although we increasingly see instances of designs that explicitly exploit properties of the environment to reduce the control overhead necessary within certain units of a certain environment.[1] Note that an unconducive environment does not rule out the possibility of certain actions to happen accidentally. This is why it is useful to complement this approach by applying in addition elements of the 'remedy' model, discussed below.

FIGURE 5. Anchor Points for Enforcement.



Next to creating an unconducive environment, one of the obviously most effective means of keeping an action from happening is to remove the affordances/tools for this action. This is the gun control model that is based on the fact that the absence of guns necessarily implies the absence of gun shots. The problem with this model lies in circumstances where a similar affordance is required in another context to which the affordance that we want to disable applies equally—and the nature of the affordance underspecifies the contexts in which it is applicable. For instance, guns can also be used for self-defence, and it is not easy to separate these two usages at the time of the acquisition of the instrument. Note that the disable class of enforcement also extends to issues through time. For example, consider a merchant who does not hand over sold merchan-

1. There is an interesting historical example of a shared computer system that exclusively relied an unconducive environment and reputation-based control for enforcement: The MIT ITS system (Incompatible Time-sharing System), an influential but highly idiosyncratic operating system written for PDP-6s and PDP-10s at MIT, was used from the 1960s to 1982 at the MIT AI Lab, without any technical access control to files. Even the 'HALT' instruction could be executed by any user at any time. The absence of control was generally considered a blessing since it meant less overhead. The fact that all users of the system shared one room provided a social context.

dise until payment for it has been secured; this transaction protocol removes an affordance for failing to pay for purchased goods.

Interrupting an initiated action if it is not supposed to happen, is another enforcement method. This is the "bullet-proof vest" case. This action-interrupt model is also the enforcement model that is commonly used in computing systems, in the form articulated by the ISO access control framework, for example. The action-interrupt model assumes that an action is to be enforced that already started to happen but that a policy might turn out to declare unauthorized, such as a user making use of a delete key to delete a file that then turns out not to be deleteable—a breakdown that then needs to be communicated specially by involving the user in some extraneous dialogue, etc.

TABLE 1. Enforcement Types: Examples and Paradigms.

Example\Type	Environment	Disable	Interrupt	Remedy	Sanction	Deter
Crime, e.g. "getting shot"	wealth distribution, etc.	gun control	bullet-proof vest	hospital	prison	police
Unauthorized file deletion	friendly atmosphere, etc.	no "delete" functionality	check predefined rights	backups & undo	loss of reputation	notification of owner
Enforcement Paradigms	Influencing Environment/ Culture	Design of Affordances	ISO Access Control Model	Design for Fail-Safe/ Recoverability	Cost Structure Design	Design for Identification and Visibility

The 'remedy' approach is based on instituting means that support one's ability of recovering previous state. For instance, rather than executing 'delete file' requests by actually deleting the object in the file system, we could have a design that always just moves such files into the "background," much like current systems that do incremental backups, but user-conceptually more integrated. Then files can be recovered and we do not have to worry as much about strictly limiting 'delete' actions to authorized users only.

Finally, enforcement can make use models that sanction or deter specific actions. Both means require appropriate mechanisms that allow for identification and authentication of a performer. In particular, an efficient form of enforcement that has been widely discussed in the economics literature is that of reputation-based community enforcement. This type of enforcement combines internal evaluation of members with the threat of being expelled from the community (cf. generally [41]). Reputation-based enforcement is a social mechanism that can be enabled by appropriate design at the technical level. Enabling mechanisms for this type of enforcement include having adequate means for identification and visibility. Interestingly, once we have these enabling mechanisms in place, we can shift some of the enforcement burden from authorization to authentication, where authentication is used to trigger a identity-based feedback mechanism that then removes the need for the system itself to enforce specific authorization policies. The examples in Section 2.4.3 will illustrate this.

2.4.2 Example: Action-Interrupt Control in the ISO Framework

Let us examine how enforcement of authorization policies is conventionally dealt with in current computing/communication systems. The ISO Access Control Framework is a representative model; it places enforcement of access control in the context of an

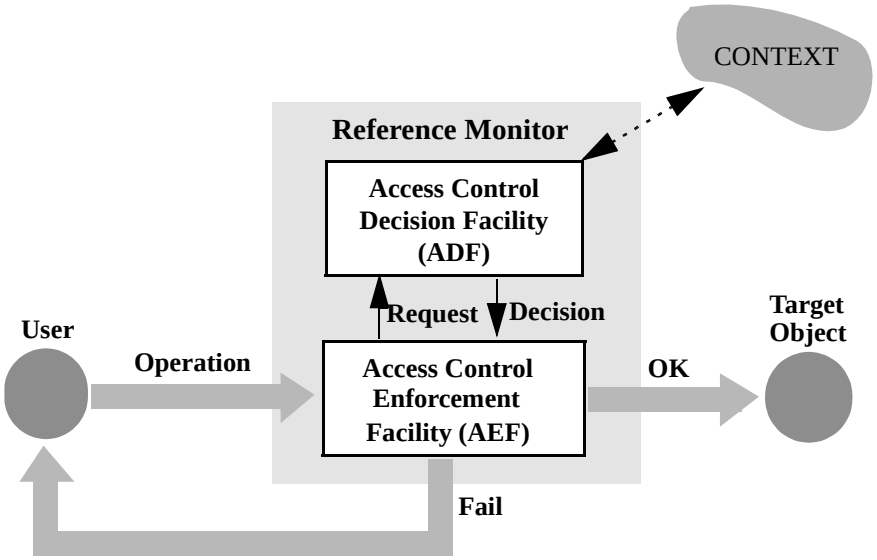
abstract technical system, and introduces relevant terminology (cf. Figure 6): Every operation is intercepted by an "Access Control Enforcement Facility" (AEF), which asks an "Access Control Decision Facility" (ADF) for a decision about whether this action is authorized. If so, the operation is performed on the target object; otherwise the AEF generates a failure exception. The reference monitor including both ADF and AEF is part of a (secure) "trusted computing base". The access-control decision facility itself is generally conceived to be based on a system of access-control rules in which specific policies are expressed (e.g., LaPadula [179]).

Note that the ISO framework is based on an action-interrupt model. From a usability perspective, the questions arise

- why a performer was led to believe in the first place that a certain action could be executed, if this action is evidently "illegal" (therefore creating the need to interrupt it to keep it from completing), and
- why no undo facility had ben put in place that would allow the uninterrupted completion of the action.

In other words, we arrive at the point where we want to shift the perspective from enforcement as rule processing to enforcement as tool-supported design of an interaction.

FIGURE 6. ISO Access Control Model: Action-Interrupt Control.



2.4.3 Shifting from Enforcement as Rule Processing to Enforcement as a Design Issue

Several examples demonstrate how technical systems can leverage forms of enforcement other than simple action-interrupt control. These examples identify the need for a design approach to enforcement that involves the design of the whole system, rather than only the authoring of rules for an ADF. In particular, they show that for many prac-

tical policies there exists no set of rules, decidable at action time, that would adequately enforce policies that people intend to have in place.

Rather than focusing on researching better ways of representing authorization rules, as done in a whole series of work; cf. [114][115][116][128][129], we therefore shift the emphasis to application-specific design. Going along with this, our interest shifts to the question of having toolkits that allow application developers to rapidly implement appropriate enforcement solutions. The commpact model constitutes a unifying core that can provide the basis for such a toolkit.

Example: University Course Material on the Web

Consider the following access control problem: A university professor uses the Web to make available material related to a class she is teaching; this includes the weekly assignments as well as their solutions. At the policy level, she has three primary concerns in mind regarding the access control for this material:

- Access should essentially be open to anyone interested. Two exceptions are however that
- students in her current class should not see any of the subsequent weeks' solutions, and
- those students who will take the same class in future years also should not have access to any of the solutions—since there might be some reuse of assignments.

Note that such an access control policy is precise, although it is not obvious how to articulate and enforce it given current systems. In particular, current models would not consider parts of it even decidable, at least not at access request time, since one does not know which students will take the class in future; in terms of a framework such as the ISO model, the policies therefore appear ill-defined.

However, here is an outline of a commpact implementation that would achieve the desired effect in a practical way. It leverages the fact that the students at her university are already bound by the student honor code. To be able to access the course material, the "communication pact" would require accessors to authenticate themselves not only minimally, but also by their full name. Note that this is not a privacy-critical application and it helps the professor to track where her efforts went. Such information would then be logged by the commpact in an appropriate form. At times, the resulting log can be matched against event masks in order to detect "interesting events," including, for instance, whether there is any overlap between students in class and accesses. If so, the professor will be notified by this event in an appropriate way.[1]

In other words, one of the requirements for our toolkit is that we have a way of communicating with the user of a certain application in a direct (application-specific) way. Note that this kind of communication channel is not even part of the overall ISO architecture.

A computational commpact object would have to be able to contain interface communication that would be able to make clear to a user that if planning to take the class (or considering the possibility), then it is inadvisable to look at the pages. Then, if a student

1. Of course, a student can always find a friend who makes available his or her account to get around the system. Note that this is like having a highly secure terminal login mechanism and then leaving the terminal itself physically unsecured.

decides to look at a certain page and to go ahead a year later to take the same class, then this event would show up on the professor's notifier and the case would have to be dealt with on an individual basis.

Example: Privacy of Medical Records

The main thing this bill does is to put everybody on notice that if they are handling sensitive patient data, they have a responsibility to that data. If they do not hold to those responsibilities, there are sanctions that come into play.
D.E. Detmer [1]

Our second example is directly taken from a proposal[1] that was considered for some time as a legislative measure. This proposal articulates access control policies for health records, and, interestingly, the way in which this is done relies on a system of creating awareness of responsibility and on making actions visible to stakeholders—rather than depending on "technical locks" to prevent access. Notification plays an essential role as a means to this end. For example, the framework provides for an obligation that law enforcement agencies would have to notify people within 30 days that they had seen their records.

Notification as an obligation takes on a function of "panoptic"[2] control. Such notification allows for the "pure" access control to be expressed more liberally (e.g., "any doctor can see the medical record"), thus reducing the need for explicit approval while still making sure that the concerned parties are aware of what is happening to information in which they have a stake.

An application of the commpact framework would deal with such a situation quite naturally. The terms and conditions of the various relationships would be represented in a set of commpacts between the different parties. Then the authorization functions that control access to certain kinds of information would make user aware of the ramifications of a certain access rather than necessarily preventing access, and implementations of objects such as notification obligations could automatically deal with the fulfillment part, say, by sending appropriate messages to the relevant stakeholders.

Example: Differential Pricing via Monitoring

A publisher would like to generate revenue from the content of its Web site. Standard possibilities include advertising (which we will not consider here) or subscriptions (which are likely to drive away many users and reduce traffic significantly).

Commpacts allow other forms of access control mechanisms to be implemented—in a form that allows material to be priced according to people's willingness to pay (differential pricing). In particular, consider the following monitoring-based implementation: Two commpacts are offered for the Web site. One of them is a "free occasional browsing" commpact that does not carry any payment obligations. This is the default commpact that users will transparently pick via their "e-persons" (cf. Chapter 5) to access the site without the need for any registration or sign-up procedure.

1. US Medical Records Confidentiality Act, sponsored by Senator Robert Bennett and Patrick Leahy, "intended to establish uniform Federal rules for the use and disclosure of health information, specifying who may see health records and under what circumstances." *The New York Times*, Nov 15, 1995, A1 ("Medical Records Are on Sale in the Marketplace").
2. A panopticon is a general-purpose architecture of visibilities, articulated first in Bentham [49], that is most widely known for its suggested application to prison design. A carefully constructed set of visibilities is used to keep people from doing certain things. In a panopticon, every action of the controlled actor is fully visible to a controller, and the fact that this visibility exists is made clear to the actor. However, it is kept invisible to the actor whether any controller is actually exercising the existing visibility, to observe any specific actions.

The only function of this commpact is to assign people a local pseudonym, to monitor their accesses, and to cause an exception in case these accesses surpass a certain frequency threshold, in which case the user is taken to be a "serious" one. Serious users will then be asked to agree to the other commpact, which might include a monthly payment obligation, for example.

Note that by using such a "loosely" enforced form of access control, we get the best of all worlds: People can freely get to know the services of a certain site, spread the word to others, link it up, etc. On the other hand, people who repeatedly care about certain services will be asked for remuneration.

Example: Shared Space of an Online Community

Online communities exhibit social boundary elements that are becoming increasingly relevant for mainstream networked environments. Such communities often share some "space," and appropriate means of access/action control for this shared resource are an issue for which the conventional techniques do not yet provide a good solution.

In this final example, we examine a typical problem in such communities: the problem of managing access to the (Internet-accessible) shared information of this community such as member profiles and interests. Several solutions are possible, each trying to find a better practical trade-off between ease of access for the "right" people, ease of keeping the information up-to-date, and security.

Conventional access control: An access control design in a conventional mindset would ask the question of who would get read/write rights to a certain profile. This leads to essentially two models, both of which turn out to be not entirely satisfactory. In the first one, elected administrators could do all the editing of the database. This is clearly puts an unnecessary load on a few people, causing overhead that will discourage updating.

The other obvious model is that of fine-grained access control based on per-user identification of accesses and corresponding item-level restrictions on editing. This will often turn out to be socially impractical for two reasons, one technical and one representational. The technical reason is that the infrastructure requirements for authentication are currently large, and there are currently only few situations where one can assume that everyone has, say, a public-key credential.

The more interesting other reason applies equally in the context of full availability of a widely adopted authentication facility with comprehensive functionality: The basic problems lies in the fact that there is usually a large amount of informal communication among the members of a community, resulting in "friends" helping each other out in editing their entries, etc. (e.g., people without Web access, people already on vacation calling back someone to turn off the mail, etc. In other words, the barrier to usability seems not to lie here in the adoption of better existing technologies, or in the fact that it is hard to make certain technologies a widely used standard, but the barrier is in the inadequate reflection of the underlying social dynamics in such formal models.

Note that the latter is conventionally treated under the label of "delegation," a mechanism whose necessity arises in the context of having introduced per-user authentication and user-based access control; delegation then essentially tries to "undo" some of the fine-grained parcellation which per-user authentication generated. An effective delegation mechanism would have to make explicit the hidden structure of social relationships corresponding to such vague notion as being "friends" with respect to editing a profile

entry at a given time, etc. Note that we do not claim that it is not possible to devise a sufficiently flexible delegation mechanism for simple tasks; we are concerned with its relative cost compared to the model described below, including the cost/likelihood of its standard availability.

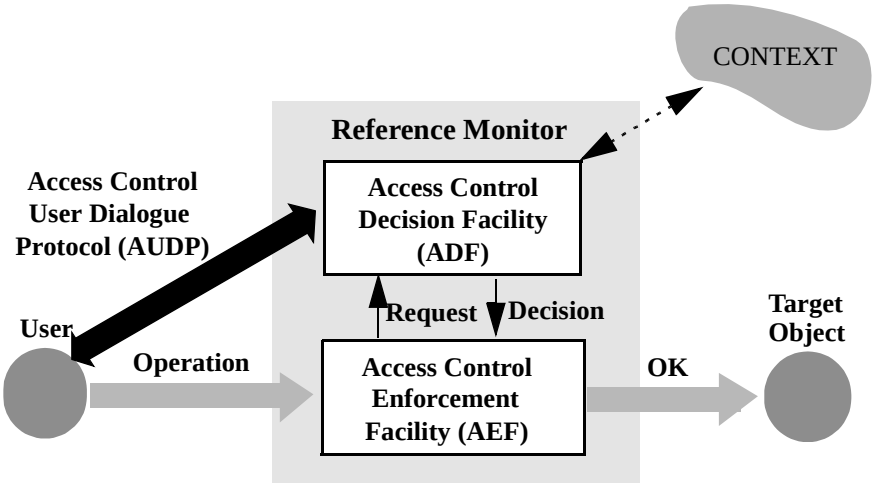
Panoptic control: We can apply a model of "panoptic" control that leverages the already existing reputation-based community control. The idea is to register stakeholders for every information item (e.g., a member's profile), and to introduce a system of visibilities (both push and pull) that will make it unlikely that anyone will create harmful modifications. Such visibilities can be in the form of making clear that there will be e-mail notifications for certain actions, or in the form of a publicly visible enriched log file that provides details about any modifications. Note that the point is not necessarily that anyone would want to look at such information, but the possibility together with the ability to get back to such information often creates reasonable enforcement of an action. Note that apart from such visibility creation, the community database can then essentially be left open for anyone (in the community) to edit it in any way—thus giving the flexibility that members can update profiles for their friends, etc.

Obviously, this setup is still vulnerable in terms of data integrity with respect to intentional or accidental deletions of entries. This can be dealt with by replication of previous state and detailed change logs (the 'remedy' model). In other words, a practical access-control solution in the case of maintaining a community database consists of a combination of group membership certification and a network of visibilities that make changes visible and assign responsibility to whoever performs the changes.

2.4.4 A Generalized Enforcement Framework

As we shall describe in more detail in the following two chapters, one of the main additions that we incorporate into the overall control model is to integrate users and their representations more fully into the access negotiation.

FIGURE 7. The Generalized Enforcement Framework: Integrating Application-Specific User Interactions into the Control Framework.



In particular, we augment the ISO model by an Access-Control User Dialogue Protocol that allows control objects such as commpacts to conduct a dialogue with users or their representations. We implement application-specific ways of exercising enforcement that appropriately reflects the circumstances of a given domain, notifying users in a user interface, etc. Figure 7 shows this revised model.

3.0 A Network-Centric Architecture for Managing Control Information

In the previous chapter, we have seen the notion of a commpact as a relationship object. In this section, we define an architecture that allows us to manage these objects in principle anywhere on the network, and we explain how this "network-centric" architecture is a natural generalization of previous architectures.

First, we will survey the three abstract ways in which control information can be organized: in a subject-centered way, an object-centered way, or a relationship-centered way. We then consider the different ways in which these structures can be distributed in a computational environment—as capabilities, access-control lists, or commpacts. The boundary conditions and design constraints are somewhat different in time-sharing environments (for which the conventional control architectures were originally designed); in client-server environments (to which they can be adapted); and in today's networked environments. In the latter, we have to deal with a multiplicity of authorities that are often at least theoretically peer-to-peer, and we also have varying degrees of trust. This creates a need for flexibly allocating control objects in a networked environment. We show how the network-centric architecture enables these usages. We use CallerID negotiations as an example usage scenario to demonstrate how communication-based transactions are more naturally dealt with by a communication-based control architecture than in conventional, client/server-based control architectures.

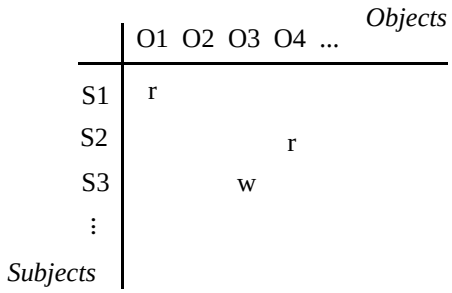
3.1 Three Ways of Organizing Control Information: Subject-based, Object-based, or Relationship-based

Control information can be organized into information structures in three abstract ways. Two of these—subject-centered and object-centered—are well-known from the "Lampson matrix" [100] that originally introduced ways of organizing control information in time-sharing environments. We revisit both of these possibilities and also lay out a third logical possibility: a relationship-centered way.

The Lampson Matrix

Let us revisit the conventional textbook description of the fundamentals of access control: Since the seminal paper of Lampson [100], it has been commonplace to view the protection problem as a large global access control matrix, where the human-organizational objects ("subjects"; matrix rows) stand in some authorization relation ("rights"; matrix entries) with information objects ("objects"; matrix columns). Cf. Figure 8. For convenience, it is also common to additionally have "groups" of people included on the subject axis, and, similarly, to have collections of objects, defined by properties, on the object axis.

FIGURE 8. Lampson Access Control Matrix.



Underlying Assumptions

Some of the *implicit assumptions* behind this Lampson matrix are the following. Note that we are not challenging the mathematical validity and usefulness of the matrix, but we want to identify sources of potential problems given the cost structure of obtaining information in networked environments.

- *Subject-Object World:* The assumption is that we have notions of "subject" and "object," that is to say, for instance, that the quality of "subjecthood" comes into being uniformly and independently of the actual interaction. We detail some ramifications of this below (cf. also Sandhu [111][112]). Note also that what is considered "object" here, is of course really something provided by another subject, the "owner," that is, the real person who is liable and responsible for it. A communication-based model would place this owner on the same level as the requesting subject. In the conceptualization of the access control matrix, owners only show up indirectly.
- *Interaction-Independent Objects:* Note that an "object" might come into existence only as part of an interaction. For example, cgi-bin scripts of Web servers can synthesize any number of objects at interaction time, without the stipulation that they necessarily exist prior to this interaction. It is possible to conceive a mathematical matrix which covers all these objects, even if this might stretch the idea of a (finite) matrix quite a bit. But this abstraction does not fully reflect the underlying real-world dynamics, and it will not be surprising then if it might not capture certain cases very well.
- *Interaction-Independent Subjects:* A similar issue holds for subjects. Clearly, we know that there are people and groups of people in the world. However, the qualities which makes them "subject" or "group" in a given context are assumed in the matrix model to be ontologically prior to the interaction between "subject" and "object." This is generally far from clear. In particular, it depends on the following assumption.
- *Open-Cards Assumption:* This is the assumption that at the point of the access control decision, all of the information that is critical for the decision is laid out "on the table." Note that this is a plausible assumption in environments, such as in time-sharing systems, where the system can take on a "Gods-eye" view. It is clearly not the case in peer-to-peer communication environments where two parties might only incrementally reveal properties that they hold, for instance, for privacy reasons. In

fact, as we will later see, for communication-based usages in the general case of content-dependent and user-dependent access control we easily run into the trouble of high negotiation cost—a fact that is partly the result of the assumption of uniform subjecthood in the Lampson matrix, which is not realized in the underlying real-world dynamics.

Abstractions that do not fully grasp the underlying dynamics often create artificial "exceptions," that is, certain cases do not fit smoothly into the framework. The notion of a "role" of a person is one example of such an exception, which arises from the fact that subjecthood is treated as ontologically prior to the interaction by which it might only come into being. "Roles" are then invented to try to fix this problem by discretizing subjecthood.

Subject-Object Conceptualizations

Having laid out the Lampson matrix, text books would then usually note that this global matrix is impractical to implement directly, and that there are two ways of realizing the abstract formulation, which correspond to the two fundamental conceptualizations that have been investigated in much detail over the past 25 years:

1. (**by column) *Object-Centered Realization*:** For each object, specify which subjects have which access rights to it.
2. **(by row) *Subject-Centered Realization:*** For each subject, specify which objects it can access with which rights.

Subject-Subject Conceptualization

Shifting to a communication model and making explicit the fact that rights are always granted by owners of objects, we see that a third logical possibility for realizing the Lampson matrix is along the lines of the relationships between the owner of an object and the holder of a right about this object (cf. Figure 9):

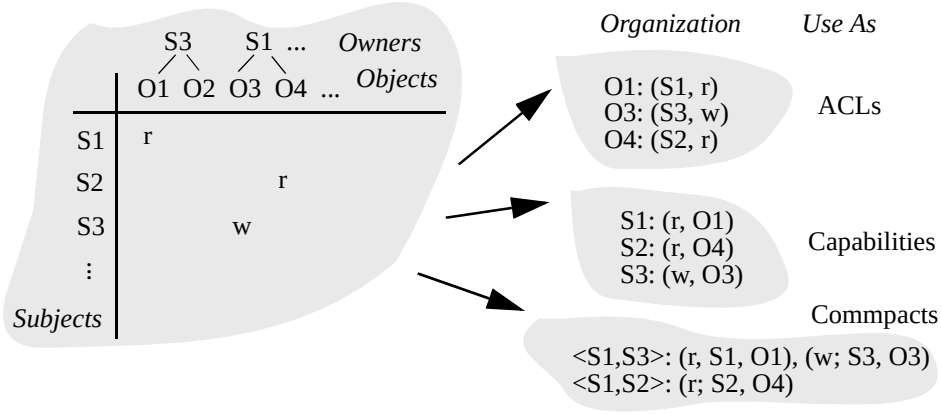
3. **(by rights relationship) *Relationship-Centered Realization*:** For each relationship between object owners and interested users ("licensees"), specify which rights each party holds about the objects covered by this relationship.

In other words, we introduce an explicit third entity, a relationship object, as shown in Figure 9. In the object-centered (ACLs) approach, there are three control objects; in the subject-centered (capabilities) approach, we get three control objects; and in the relationship-centered (commpacts) approach, we have two control objects in this example (one for each relationship).[1]

So far we have only considered the abstract organizations of control information into objects; we have not yet examined the orthogonal question where to place such objects.

1. The example happens to create fewer objects for the relationship-based realization of this typically sparse matrix than for the other ones. We do not wish to claim that this is necessarily a general property of this realization though. In general, the relationship-based form will have a smaller number of control objects if a lot of objects are governed by a few relationships.

FIGURE 9. Realizations of the Lampson Matrix (Revised): Organizations and Uses.



3.2 Three Ways of Embedding Control Objects

Information can be placed and used in a network at three different types of locations: clients, servers, or anywhere else on the network. Depending on how we distribute control information, we therefore obtain different models of control management. The object-based realization lends itself efficiently to use as server-based access-control lists (ACLs), the subject-based organization as client-based capabilities, and the relationship-based organization as commpacts. Note that commpacts are the only structure that is symmetric and that is therefore not bound to a specific location.

In the time-sharing environments for which the earliest designs were developed, implementers had homogeneous control over the security and trustworthiness of the different parts of the system. It was therefore possible to easily associate capabilities with requesters or ACLs with the accessed objects.

In client/server environments, this architecture generalizes in a fairly straightforward way; the only difference is now that control information associated with subjects cannot be trusted quite as much, and we therefore need to authenticate the asserted capabilities properly, say, by using public-key tokens. Note that this is an authentication of the capability information itself, apart from the authentication of the subject, which we also already have in time-sharing environments of course. Also note that a combination of the ACLs and capabilities, a “lock-key” mechanism, is often used in practice: At first, ACLs are used to determine rights; then these rights are then “compiled” into an access capability. However, this is really mostly an implementation optimization.

Once we move to a networked environment, we have additional boundary conditions for a control design:

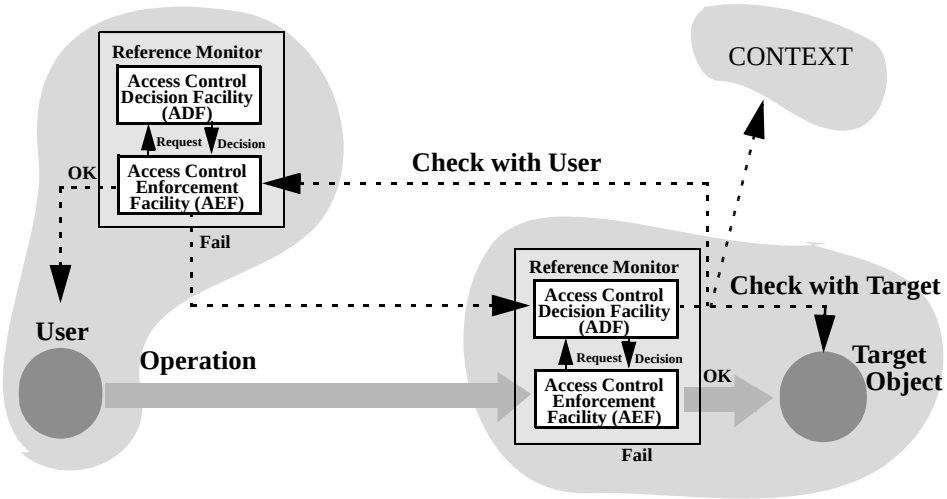
- *multiplicity of authorities and trust levels:* There is a multiplicity of social authorities that have a stake in the representation of various control structures, and many different trust preferences need to be accommodated.
- *peer-to-peer nature:* The participants in the environment are peer-to-peer rather than “supplicant-granter”; at least there is such an equality conceptually even if differences in bargaining power do clearly exist in practice.

In such an environment, the issue of where to place control information becomes more critical. In particular, a symmetric control structure such as commpacts—which would have been unneeded overhead both in time-sharing environments and in client-server environments—become a useful representation. Commpacts allow us to place control information where it fits given the trust preferences of the relevant stakeholders.

3.3 Understanding Conventional Control Architectures

In this section, we lay out how conventional control architectures that stem from a client-server world are ill-suited for peer-to-peer networked access control. We demonstrate how the negotiation cost can easily get high in such conventional architectures for usages that are inherently subject-subject communication applications.

FIGURE 10. Authorization Interactions: Decision Facility Requests Attributes.



Let us reconsider the general access control architecture defined in the ISO framework as discussed in Section 2.4. As we noted there, the basic model is to have a reference monitor consisting of an enforcement and a decision module that make sure that only authorized actions can go through without interruption. The authorization decision is based on evaluating a set of access-control rules that will in turn depend on properties of the system context (e.g., the time), the target object, and the requester. An example for such a general policy which we might want to accommodate could be "Approve all requests from US citizens for documents which have not been modified since last week."

Requests from the target's trusted reference monitor to the user for confirmation of user attributes ("Check with User" in Figure 10) would be intercepted by access control again—this time by the user's trusted reference monitor. Note that, in a distributed setting, each participant has their own authority to determine by which rules they wish to participate in the system, and different users' trust preferences will therefore generally not extend to the same reference monitors.

In this general case, the reference monitor takes on more of the role of a "negotiator" between client and target. Although simplifications are possible in the case where cer-

tain entities are fully trusted, in general, the rule interactions within a given reference monitor and those between different reference monitors (user-trusted, target-trusted) are less than obvious, and the negotiation costs can easily get high. Not only do the access rules within one system have to be appropriate, but they also have to work together in the right "incremental revelation" schedule with those of the rules protecting other objects.

For example, in the nationality-based policy mentioned above, the requester has to understand that when claiming access with respect to this policy, then the otherwise private nationality attribute must be revealed to the target's reference monitor. Such interactions can become complex and difficult to understand, and Moffet and Sloman [113] conclude that such general, application-independent access control will therefore not be practical.

Example: Negotiation Cost for Simple CallerID Interactions

Let us consider as a simple demonstration example a set of rules by which people might want to determine under which circumstances others can call them on the phone. The privacy implications of such access rules have been extensively debated under the name "CallerID". In this case, much can be resolved by going to a general access-control system which enables participants to articulate the conditions under which they are willing to participate in a communication exchange.

Consider the two communication participants Tom and Lisa, each of whom expresses preferences in a set of access-control rules (cf. the Datalog-like pseudocode in Figure 11). Each person has a set of attributes such as `name`, `ID`, and `callType`, which are communicated only when the corresponding `reveal` access predicate allows it. The phone bell is accessed here by the function `connect_call`, which determines whether or not the call will be connected. A notation of `A.name` is used to access the name attribute of `A`; if this is executed by someone other than `A`, then `A` is asked to reveal this attribute.[1] Specifically, `reveal` is a special predicate about a personal information attribute; if there is a rule which makes it true, then the corresponding attribute is returned.

1. There is a certain body of work in distributed logic programming etc. which is looking into how to transform rule systems in order to minimize communication overhead (e.g., Wolfson and Silberschatz [233], Saraswat *et al.* [234]). However, these works generally do not consider constraints pertaining to boundaries of authority/ownership and privacy of the locally owned rules, that is, limitations as to which processors can be trusted for what.

FIGURE 11. CallerID Example: Simple Set of Phone-Access Rules.

Person A: Tom

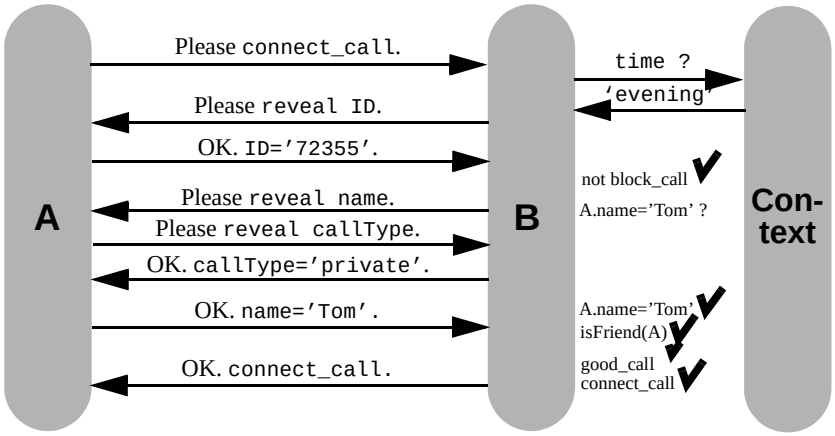
```
name='Tom'.
ID='72355'.

reveal(name) IF
  B.callType='private'.

reveal(ID).
```

Person B: Lisa

```
name='Lisa'.
callType='private'.

reveal(name) IF isFriend(A).

reveal(callType).

connect_call IF
  NOT block_call AND good_call.

block_call IF
  Context.time='evening' AND
  (A.ID='6953' OR NOT A.ID).

good_call IF isFriend(A).

isFriend(A) IF A.name='Tom'.
```

Figure 12, shown below, lays out the temporal sequence (top-down) of the authorization interactions when A calls B in the evening. Notice the brittleness of the system: With the distributed rule authorities, bugs can easily be introduced by one party by not sufficiently considering the possible dynamics which might result from unexpected interactions with the unknown policies at other sites. Indeed, in the general case, not even the possibility of deadlocks can be ruled out. Not only is the negotiation cost high, but also the usability of such a system is likely to be low.

FIGURE 12. CallerID Example: 'A calling B' Leads to Complex Negotiation.



The underlying reason for this is of course that we have here a coordination problem which needs a language shared among the participants; if such a language provides high-level primitives, then the lower-level transactions which account for much of the negotiation cost can be avoided. Note that A does not know a priori what B wants to know, and vice versa. Moreover, privacy considerations dictate that only as much information as needed to lead to success should be released.

The conceptualization which we will suggest as an access control framework is targeted at avoiding this negotiation complexity by bridging the gap between requester and target with an intermediate concept, which encapsulates interdependent access control policies. This would reduce negotiation cost and might enable general access control policies.

3.4 From Server-Based and Client-Based Control to Network-Centric Control

In this section, we describe a control framework that

- puts control information as first-class objects onto the network,
- encapsulates interdependencies on a per-relationship basis, using a (peer-to-peer) communication model rather than a client-server model,

First-Class Control Objects

The first step is that rather than attaching control information to controlled information, the model is that we encapsulate control information into first-class control objects that designate the objects that they control (by using a constraint). In this way, we keep independent two dimensions that are orthogonal: the question of which controls apply and the question of which objects control is applied to.[1] Encapsulating related control information helps us to factor out unintended rule interactions and supersede some of the negotiation cost problem, as we saw it in the previous section.

Introducing a Network API for Control Requests

The next step is then to allow the control objects to reside in principle anywhere on the network—which is obviously why we call it a "network-centric" design. Figure 13 depicts the new control architecture. Effectively, the Access-Control Decision Facility is moved onto the network, and there is a standard API introduced for requests to it.

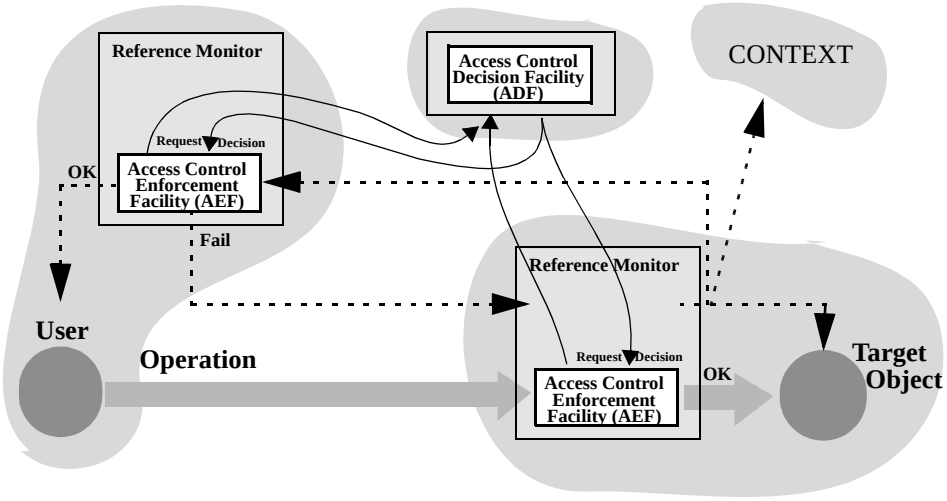
Relationship-Based Control

The third step is to organize the policies contained in the Access-Control Decision Facility in a relationship-based way, and distribute the control information in a way that reflects the trust and usage pattern that characterize a given relationship. We want to reconceptualize control information along the lines of relationships. Relationship objects (commpacts) are then the baseline with respect to which authorization is done, and we can place them in locations with matching trust expectations. Figure 13 illustrates how commpact objects have an interface that includes authorization functions that are typically performed by an Access-Control Decision Facility.

Commpacts are then effectively a network-centric variant of an ADF. Actions are authorized by using the commpact that the actor designated as the baseline with respect to which a certain action is to be performed. The choice of the appropriate commpact will usually be determined by preference rules in a way that is transparent to the user. The number of commpacts that a person can have is unlimited in principle, although specific implementations will have resource limitations, of course. There is one commpact for every relationship.

1. Note that, strictly speaking, it is therefore incorrect to say that "rights languages *attach* control information to objects"—since they just need to set it in relation with each other. Attachment is primarily a matter of the delivery mechanism (NNTP, etc.), not a matter of the control design.

FIGURE 13. Network-Centric Control Architecture.



Network-Centric Architecture

With commpacts residing in principle anywhere on the network, we need an architecture for managing them. We call the object managers that manage commpact objects simply "commpact managers." Commpact managers are servers that manage commpact objects. In other words, the idea is to have a set of professionally managed services that deal with control information in a secure and reliable way.

In the network-centric architecture, commpacts stand in a *m:n*-relationship with the objects (and services) that they control. Trust management is used to determine which objects accept control by which commpacts. For example, a specific network service might only trust authorizations from a certain commpact manager. Furthermore, a constraint determines which objects a certain commpact controls.

The network-centric design generalizes the other models of organizing control information: the client-centered, possession-based capabilities model and the server-centered model. While commpacts can be co-located with the information they control (at a server), they do not have to be so; for instance, they might as well just reside with a third party, such as a rights clearing house. However, we can still consider it as the default case that commpacts will in fact actually just reside with the server of the controlled object, as in conventional access control. The main point in the network-centric architecture is that we have the protocols and APIs that give us structured access to relationship state, independent of the location of this information. This provides the basis for achieving the kind of end-to-end integration between client applications and network services that we have been targeting. It also gives us the flexibility to independently instantiate and modify commpacts and the objects that they control. For example, a publisher could provide a pay-per-view contract, and at some later point in addition a subscription agreement for the same online content.

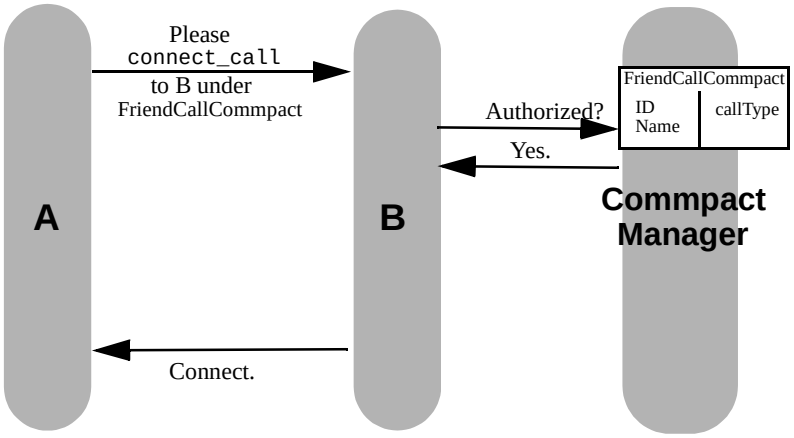
In other words, in the network-centric architecture, we augment network services by a network-based authorization facility that functions much in the same way as authoriza-

tion is currently done, but that also provides a principled way of exchanging, managing, and interacting with such control information.

CallerID Example Revisited

Communication-based usage scenarios can be easily dealt with in a commpact control architecture. Here is how the CallerID interactions in Figure 14 would look in the communication agreement framework.

FIGURE 14. CallerID Example: ‘A calling B’ with commpacts.



We have to distinguish two cases:

- *Normal Case:* The two parties are already in a relationship; a commpact exists. The commpact is referenced by the caller (possibly transparently to the calling person), and used to authorize the action.
- *Negotiation Case:* No previously negotiated commpact exists. In this case, a new commpact that contains the shared control information for this relationship would be negotiated between the two parties, possibly automatically by using preferences articulated by the two participants.

Note that a number of special cases which have been raised as objections against the introduction of Calling-Number identification can be readily dealt with. For example, it was pointed out in the debates surrounding the CallerID issue that

- certain people like psychiatrists might want to call patients without revealing their number. This points to the necessity of a blocking feature for some circumstances.
- if someone calls 911 in case of emergency, then the blocking feature, which might have been enabled, should be ineffective since otherwise the caller cannot be located.

In the commpact model, each of these types of behaviors would basically get a different commpact. Next to the commpacts mentioned above, there might simply be an `EmergencyCallCommpact` for calls to 911, and an `PatientCallCommpact` for cases such as the one mentioned above.

3.5 Security Implications

Having commpacts reside on the network, managed by specialized commpact manager servers, clearly has a number of security implications, at least in the general case. As a first issue, since it is a "network-centric" design, we obviously depend on the availability of the network. Without reliable network access, this architecture will not live up to its full potential. For usages that we have targeted, the assumption of network access is something that we are willing to take.

Once we get beyond the network availability issue, we have a twofold situation: If commpacts are just used to provide structured access to control information (via the APIs/ protocols that they define), and if they are managed exclusively in a server-side way, then no additional security considerations are implied beyond what we already have in current systems. For instance, if we run a commpact manager within the same immediate trusted computing base (e.g. the same machine) as the network service that we wish to access-control, and if we tightly control who can instantiate new commpacts with this commpact manager (e.g. no outside person), then we have no new security implications.

As soon as a service uses a commpact for authorization that is not located within the same immediate trusted computing base as the service itself, we need a mechanism to determine whether this outside commpact is trusted: that is, whether this commpact belongs to our dynamically extended trusted computing base. In particular, we have to assure that no "bogus" commpact is used to authorize actions that we care about. Several mechanisms exist to dynamically extend a trusted computing base for such purposes. Parts of the system in Stefik [97] are about exactly this question, using a protocol that extends existing communication protocols by a prologue exchange where communicating systems make sure that they trust each other by demonstrating public-key credentials to each other.

For our specific purposes, the following scheme can be used by a service that wants to determine whether it can trust an authorization from a commpact that a requester wants to use for the service. Every service has one or more providers ("owners") that have the right to give others access to it. Network services maintain a list of their providers. The problem is then for the service to determine whether a given commpact was agreed to by one of its providers. Since services know their providers, standard cryptographic means can be used to make sure that we are not dealing with an invalid commpact: Services only trust a commpact's authorization if the commpact can provide a cryptographically signed token that shows that (one of) the service's provider(s) agreed to this commpact. The commpact itself would get hold of this token during negotiation.

Note, however, that the scheme in the previous paragraph is only necessary if we cannot establish the presence of a trusted computing base through simpler means, such as using the fact that a service is run as part of an Intranet, that it is run on the trusted commpact manager of a partnering organization, etc.

3.6 Linking from Content Objects to Rights-Management Information

As we have seen in the previous sections, commpacts are managed independently of the objects that they control. Network services maintain a list of their providers such that they can determine which commpacts they can trust for authorization purposes. However, commpacts can control not only network services, such as a Web search engine, but also content objects, such as the issues of a subscription-based online newsletter.

For content objects, several delivery mechanisms exist, including SMTP, NNTP, and HTTP; and a variety of packaging formats are used, including MIME and its subtypes as well as various content container technologies [96][93][143][145]. The question is how the interface between content objects and commpacts is defined. Specifically, the question is how content objects can link to their provider(s). As discussed in the previous section, all the other aspects are just the same for content objects as for general services.

Commpacts complement secure content container technologies in that they use them as one of the access-control enforcement facilities from which authorization requests might originate. Content containers provide a mechanism by which accesses can be interrupted to make sure that the request is authorized—in our case from a requester-designated commpact.

In order to make the commpact scheme work for content objects delivered by standard delivery protocols, we need to point to the provider(s) of a content object as part of its metadata in its header or in its content container. Note that we do not need anything else, since commpacts are independently managed.

Incidentally, the Dublin Core attribute model [270], for instance, already defines a metadata standard for documents in which rights-management information is intentionally kept outside the scope of the standard, except for a pointer to rights management information.[1] This fits well with the commpact model in that we can use this rights-management attribute to provide a reference to the content object's provider (owner). For e-mail and newsgroup articles, we can use document header information such as

```
X-Provider: martin@Epersons.Stanford.EDU
```

for the same purpose. Using a name server, an object name can be resolved to find a corresponding object handle for the provider's "e-person" representation (cf. Chapter 4 for further details). From there, we can then request the set of commpacts that are available for the object. In a distributed object implementation, the name server of this object system would be used.

Alternatively, we could also directly include a list of offers as part of the metadata:

```
X-Offers: [PayPerViewLicense-34@Licenses.Stanford.EDU]
```

Such a list would point to corresponding commpact offer objects, that can be found at some commpact manager.

In summary, the impact of the commpact framework on existing content delivery and packaging mechanisms is minimal. Existing mechanisms meld well with the use of commpacts.

3.7 Related Architectures

Relationship-based control is not the same as what we called client/capabilities-based control. [95] Capabilities are opaque tokens that reference a relationship context that itself might still be predominantly defined elsewhere: While the client might possess the

1. The Dublin Core defines a basic set of attributes for documents, such as 'title' and 'author'.

token, the interpretation context within which this token is meaningful is usually on the server.

From a user's view, capabilities usually appear as the familiar "tickets"—with the same set of associated problems: when they are lost, they are gone; trying to revoke any of them guarantees to be a major enterprise, etc. Unlike full-fledged contracts, tickets only provide limited information about the context within which their use came about—which is why they are used almost exclusively in situations that are either characterized by low stakes and a strong imbalance of trust/bargaining power (e.g., Joe Individual vs. National Railway Operations) or low stakes and immediate fulfillment (e.g., movie theater tickets). For such special cases, tickets are a low-cost variant of externalizing contract information. Some technical systems (e.g. [96]) take this as a reason to only reify tickets and abstain from reifying contracts themselves. While certainly a plausible trade-off for many applications, this limits the affordances available for users. For example, while they might be able to purchase a ticket and use it to gain access to online content, there would be no structured ways of cancelling the contract and returning the ticket, or just obtaining information about such things as which warranties a good has.

Client-based access control as implemented by SmartCards has been known to have advantages in cases where a policy requires various forms of strings attached, say, in the form of client-side actions that need to be controlled, such as limiting the time of usage. In such cases, we need to make sure that the client has the control information at hand at any point of time—even after an initially positive access authorization.

4.0 FIRM: An Infrastructure for Digital Relationship Management

In this chapter, we take the next step towards an implementation model and describe the kinds of objects and transactions that we can use to realize the commpact framework. In particular, we will describe the Stanford Framework for Interoperable Rights Management (FIRM).

FIRM defines a relationship-based rights management service layer on top of existing Internet protocols, supporting a host of usages including, but not limited to, digital contracting, privacy negotiations, and network security. These usages can be enabled in an incremental, bottom-up way since FIRM is designed to be able to evolve naturally from services present in the current Internet. In particular, FIRM implies an institutional context that does not require the development of any new institutions such as a "Digital Property Trust" (Stefik [97]) or a centralized authority to enable the management of property rights (IBM [143][144]). FIRM's centrality properties are only at the level of organizations (via services such as "home providers" and "forms designers," as we define them in this chapter) rather than at the level of the complete networked environment, where we only assume the adoption of a generic interface standard.

FIRM has been prototyped as part of the Stanford Digital Libraries Project in a system called RManage, a prototype relationship manager application. In this chapter, we will draw on examples of RManage to illustrate the FIRM network service layer.

Sections 4.1 and 4.2 describe FIRM's object reifications and its transaction model, respectively. This is followed by a survey of the kinds of user interface affordances that a FIRM implementation can provide. In particular, we will draw on examples from the RManage prototype implementation for this purpose. In Section 4.4, we will then give a sample transaction scenario that describes how the various objects can interact. Finally, in Section 4.5, we will describe related work. Further explanations of FIRM including a formal object-request interface specification can be found in the appendix of this thesis.

4.1 Object Reifications

In this section, we describe how we computationally reify objects including persons, and roles of persons, agreements, agreement forms, promises, and constraints, as well as the object managers that manage each of the reified objects. The latter include online "home providers," "forms providers," and "relationship managers." The following is a summary of some of the major object reifications in FIRM. See also Figure 15.

E-Person: An e-person is a software agent that is a persistent online representation of a person, or of one of its roles. E-persons have a structured request interface that allows clients to request approval, negotiate access conditions, access personal context information, etc.

Home Provider: A home provider is the network service that manages e-person objects and makes sure that they are constantly available for network requests.

Commpact: A commpact is the computational object that is the digital representation of an agreement between two or more parties, be it a legal contract or a more lightweight "communication pact" (e.g., one related to privacy). It is a "smart contract" in

that it is based on code that can generate descriptions about its current state, enforce some of the terms and conditions, etc.

Commpact Manager: A commpact manager is the network service that keeps, manages, and interprets commpact objects that have been assigned to it.

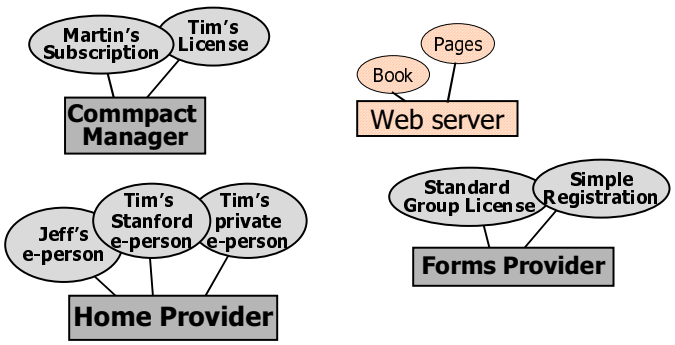
Commpact Form: A commpact form is the basic template of a commpact. Commpact forms are the "stationery" that people can customize, fill out, and then offer.

Commpact Forms Provider: A forms provider is the server that makes available a collection of commpact forms.

The basic scheme is that form designers develop standard digital contract forms. These are object implementations that can describe their state, that can automatically enforce at least some of the policies, etc. These forms are then made available as "stationery" that anyone can take, customize, and instantiate. People are represented online in the form of an e-person. An e-person is hosted by a home provider; it has its various contractual relationships (commpacts) managed by any of a number of commpact managers. The latter can reside anywhere on the network, whether at a server (conventional access control), at a client (usage control), or with a third party (e.g., a rights clearing house).

Note that some of the terms either are new words or they are phrases used with a special intended meaning; we introduce new words such as 'commpact' and 'e-person' to distinguish between real-world objects, such as a legal contract or a real person, and their electronic representation (the digital objects).

FIGURE 15. FIRM Object Reifications: Commpacts and commpact manager, e-persons and home provider, forms and forms provider.



4.1.1 Reifying (Roles of) Persons: "E-Persons"

E-Person: *An e-person is a software agent that is the persistent online representation of (a role of) a person with a structured request interface. When acting online, users are identified by a (possibly opaque) handle to their e-person, allowing any communication clients to talk back to this structured representation. E-persons make it possible to have certain kinds of transactions take place on behalf of a user without that this user necessarily needs to be involved in it directly. Users can set up default preferences that determine the actions that the e-person might automatically execute. This includes*

negotiating access conditions, for example. A Unix account can be seen as a current form of a limited version of an e-person.

E-persons are a way of conceptually separating out the reality of a physical person and the image created by its online behavior. An e-person essentially can be seen as a more structured generalization of existing variants of this concept, including controlled ways of getting hold of personal information, requesting (automatic) approvals, leaving behind notifications, automatically setting up certain standard relationships such as accounts with content providers, etc.

An e-person has generic (access-controlled) interfaces for *information push* (e.g., receiving e-mail, other notifications, etc.) and *information pull* (e.g., requests for personal information, polling for preferences, etc.). Every e-person is also assumed to have a "notifier," that is an inbox that provides a uniform way of seeing "what's new" in a way that accommodates a structured set of actions and that also equally extends to information push and pull. E-persons can enter relationships with other e-persons by agreeing on commpacts.

One and the same person can have multiple e-persons; for instance, a certain individual might have one e-person for private and one for business. If a physical person has more than one e-person, one of them can be designated to be the person's super-e-person, that is, the e-person from which the others "inherit" basic properties of the person, in a classless inheritance scheme. Inheritance means in this case that one object will proxy requests to another object for a defined set of its properties.

Example: Bob has access to three e-persons, one for him as a private person, one for him in his role as treasurer of an organization, and one for him as a partner of his firm. His firm agrees to be home provider for all three of them, but, for the private one, it grants only limited disk space and for the treasurer one it has disclaimers about any liabilities that might result from it. Bob designated his private e-person to be his super-e-person. His birth date is therefore stored with his private e-person, but in principle available to the others via inheritance.

E-Persons as User Agents, Enabling a Network Login

The two main functions of an e-person are to act as an agent and to enable a network login. E-persons represent users online and act on their behalf for certain standard interactions. For instance, e-persons allows users to access information by providing the appropriate passwords that might be necessary for a service, or possibly even by registering users automatically or by accepting certain contract offers automatically—if this is within the space of what user-defined preference rules endorse. This allows us to reduce the transaction costs inherent in a contracting scheme and to keep as many interactions as possible out of the face of the human user.

People have a way of authenticating themselves (the programs they use) with respect to one or more of their e-persons. The authentication step itself can be conducted using any of a variety of schemes, including Kerberos authentication [207] or Unix-style password comparisons [171]; but we assume that as a result of such an authentication, clients will obtain an authentication token that can be presented to any server such that the server

can use it to confirm the requester's identity.[1] This token can just be the name of the user's chosen e-person and the network address of the client program, both signed cryptographically by the e-person's home provider. Any server can use the home provider's public key to make sure that the program that sent a request from a certain network address is in fact associated with the e-person with which it claims to be associated.

Once authenticated by a *network login*, the conceptual assumption is that an e-person is identified by its epersID for every action that it (or one of its user's programs) performs. Being identified by an epersID is much like being in a room with other people without knowing their name or any other attributes; one can "address" others and try to find out more about them as part of a negotiation, but by default nothing will be known about an identity. In this sense, an epersID can provide a quasi-anonymous identification if the user chooses as a policy to deny inquiries for further personal information.

Note that various limited forms of an e-person currently already exist, albeit disparate and not uniform. For example, a standard Unix account can be seen as a preliminary form of an e-person, limited to a simple information push interface (adding e-mail messages to its mailbox) and an information pull interface in the form of a Web home page or a directory server entry. Note that the notifier in this example takes on the form of a disparate set of units, including a user's mail inbox, a news inbox for unread newsgroups, as well as a variety of notification events in applications such as calendar programs. The epersID for an account would be simply that person's e-mail address. (See the Grassroots system [279] for a more extensive use of the notion of a notifier.)

In the RManage prototype, e-persons are CORBA objects with the request interface given in the Appendix. User's client applications convey the string binding handle (or a name that can be resolved by a naming service) of their person object to any server that a user might be talking to; this handle is part of a public-key token that is signed by the e-person's home provider. Augmented servers can then use this information to authenticate the requester—that is, verify that a request from a certain network address is indeed from the e-person object whose handle was provided—and talk back to the requester's representation about further details. New client programs can directly submit the epersID token; for Web browsers, HTTP cookies are used to send along this information with every request. EpersID tokens have an expiration time; they are initialized at network login time.

4.1.2 Managing E-Persons: "Home Providers"

Home Provider: *A home provider is a network service that manages e-person objects and makes sure that they are always available for requests. It thus provides an "online home" for persons (in the form of an e-person). Home providers can be seen as a value-added extension of current online services and ISPs.*

1. If the client application is a Web browser, then HTTP cookies can be used to convey this token to servers. This implementation works well for a limited number of FIRM-enabled services, as in the RManage prototype. However, at this point, this approach would not work for full-scale use due to the fact that most browsers currently do not allow HTTP cookies to be accessible to all servers; cookies are required to be limited to a specific set of servers. Client-side proxies could be used to get around this problem, but in the case of full-scale use, one probably would want to incorporate this mechanism more tightly into the HTTP protocol.

E-person objects are persistent representations of persons in the online environment, even if their users are not "logged on" at a given time. They have to be provided by some institution that has the resources to keep the person objects reliably running all the time. We call this object manager a "home provider."

As mentioned in the previous subsection, by making available the abstraction of an e-person, home providers can serve as privacy intermediaries for their members and control access to their personal information and their attention or time. This includes support in securing proper authentication of their members without requiring global identities and consumer-side public keys.

Current online services and Internet Service Providers can be seen as examples of preliminary versions of such home providers for consumers on the Internet. Universities and companies currently provide similar in-house services for students and employees; each of these can be seen functionally as an instance of a home provider. But the notion of a home provider also extends to other domains. In the case of electronic trading (using EDI standards; cf. [161] for an introduction), the "EDI network provider" fulfills functions that we attribute more generally to a home provider. These include reputation-based management of membership, authenticating members, and certifying user attributes. Generally, home providers will have service contracts with their members that allow them to regulate which kinds of electronic activities are binding under which terms and conditions, and to which rules their members are committed for interactions among themselves.

4.1.3 Reifying Relationships/Agreements: "Commpacts"

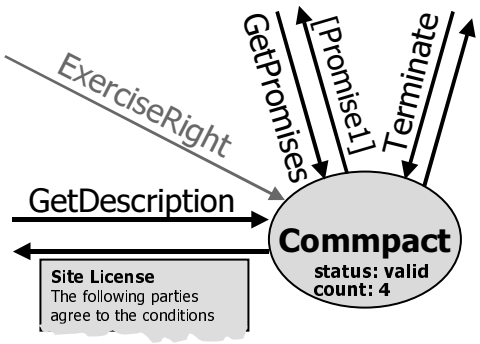
Commpact: *A commpact is the "relationship object" that is the digital representation of the agreed-upon terms and conditions of the relationship between two or more parties, be it a legal contract or a more light-weight articulation of a "communication pact." Commpacts are "smart contracts" in that they have a structured (FIRM) interface, code that implements behavior, state, and a set of textual descriptions. Commpacts contain a mixture of informal textual descriptions and implementation code. The fact that both have the same semantics is the responsibility of the designer of the underlying commpact form.*

A "commpact" is the electronic representation of the "communication pact" that encapsulates the boundary conditions of a relationship between two or more communication participants. A major subset of such communication pacts has externalizations in the legal world in the form of legal contracts. Other commpacts have quite different externalizations and associated enforcement means, such as social conventions enforced by community reputation-based enforcement. For example, anonymous ftp on the Internet is conceptually a commpact that says that anyone who identifies herself with her e-mail address can access the public directories of the file archive and download material.

Architecturally, commpacts are first-class objects that are managed independently of the objects about which they provide control behavior. Commpacts *designate* the objects about which they are in the form of a constraint about objects. For example, a subscription contract for a year's worth of newspaper issues will be represented by a commpact containing a constraint that designates all the objects that are by a certain publisher and that have a publication date in a certain range. (Note that some of these issues might not even exist yet.)

Commpacts have a structure that is formally specified by the APIs defined by FIRM (cf. the Appendix). Commpacts authorize actions, enforce prerequisites, and provide a way to live up to obligations, such as by initiating a payment transfer. The piece of text by which we generally know legal contracts is just the result of one of the many methods that can be called on commpact objects–but there are also others, including negotiation methods (e.g. 'terminate'), structural messages ('get me the set of promise objects'), and, last but not least, authorization interactions ('exercise this right'). In other words, commpacts are objects that have a structured request interface (as defined by FIRM), code that implements their behavior, state, and a set of textual descriptions from which descriptive information can be generated. Cf. Figure 16.

FIGURE 16. Commpacts as "Smart Contract" Objects: FIRM Interface + Code + State + Texts.



Commpacts have a method by which one can obtain a human-readable description about them. These descriptions are generated dynamically based on the attributes that commpact objects are defined to have (by the FIRM specification). For instance, in our prototype system RManage, such a description is an HTML page that describes the terms and conditions of a subscription contract with links to descriptions of the terms and conditions of the access right and the payment obligation that are part of this contract. It is the commpact forms designer who makes sure that the textual descriptions generated by a commpact object reflect the meaning of the underlying structured representation.

Commpacts also contain machine-interpretable code that automates parts of their behavior. In other words, they not only articulate certain promises, but can also enforce at least some of the terms and conditions of the agreement.

The extent to which contract policies are implemented depends on the extent to which corresponding infrastructure is available online. For example, various regulations about dispute resolution in courts will plausibly just be part of the general terms and conditions of the agreement; however, in the hypothetical case where online infrastructure is available, say, to automatically register complaints with a court, such behavior can be taken up into the code implementation of the commpact—rather than merely in the textual descriptions.

Commpacts can be interpreted at any of a number of trusted interpreters ("commpact managers"), including interpreters run by (and presumably co-located with) the content server (conventional "access" control), the client machine (for high interactivity or for

the mobile case), or a third party such as any kinds of “rights clearinghouses” (copyright clearance centers such as CCC, IVY, etc.).

Reifying Contract Law

FIRM closely follows contract law principles [1][2][3] in defining the structure and behavior of its objects. A FIRM interface specification is available in CORBA’s Interface Definition Language [5], although nothing in FIRM intrinsically relies on the availability of a CORBA distributed object infrastructure (see the Appendix of this thesis).

At the most elementary level, *commpacts* are just a set of enforceable promises between two or more parties. *Promises* in turn are either rights or obligations. *Rights* allow their holders to do something, *obligations* require them to do something. For example, the obligation to pay a certain fee for a subscription to a newsletter might be one such promise as part of a subscription contract. This obligation object could keep track of the size of the outstanding obligation, the past payment history, etc. It might also directly provide means to initiate an automatic payment transaction, say, just before the due date.

Each *party* participates in an agreement as part of a certain agreement role. *Roles* are characterized by a constraint on who can fill the role. There can be two or more roles for one agreement, and each role except for the ‘offeror’ role can possibly be filled by more than one party (instance). A commpact has an ‘AboutItems’ constraint that characterizes which items the agreement is about. In other words, the control object quantifies over the objects it controls rather than being attached to these objects.

Promises have components such as a promissory condition (‘ConditionsPrecedent’) that specifies which conditions need to be in place before the promise becomes effective, a ‘ConditionsSubsequent’, and a constraint that specifies the objects that it is about (unless the promise is about the same objects as the commpact). Promises can be waived, transferred, etc.—where transferring an obligation is different from just allowing someone else to fulfill it; in the former but not in the latter case, it becomes embedded into a new contract context.

Each FIRM object has a method by which one can obtain a description about it and a reference to it. The reference includes a persistent name. It might also include the current object address, say, for a distributed object, but a persistent name is always provided. If applicable, this name can then be resolved into an object address, using the name server of the object system employed.

Using the Access-Control User Dialogue Protocol (ACUDP), users can directly interact with commpacts, as well as other FIRM objects, for such purposes as customizing parameters, accepting offers, etc. In the simplest case, this protocol is just based on HTTP using the standard HTML forms/URL-encoded parameters convention. Further detail about this can be found in the Appendix.

Not Everything is Reified: Two Examples

While contract law provides the basic conceptual framework for how to represent objects, not everything is carried over exactly as it is in law. In particular, in legal frameworks, we find various provisions that are geared towards making the framework work more smoothly in the real world, say, by taking into account some of the obvious shortcuts that are likely to be made by people given the cost.

For example, there is a notion of "silent acceptance," by which an offer can be accepted without having to explicitly issue an acceptance—but thus also creating the necessity to explicitly reject an unwanted offer that might otherwise qualify for silent acceptance. Another example of such a transaction cost issue is the concept of a "unilateral contract" (a promise given in exchange for an act, such as "I pay you $100 if you paint my house"), where a mere fulfillment action would already bind the promising party even if no separate, explicit acceptance action ever took place.

We have usually chosen not to reify such cases in our digital representation since the cost of executing an additional protocol action is relatively minor in an electronic infrastructure relative to the cost of the additional ambiguity that is created by relying on such short-cuts.

4.1.4 Managing Commpacts for an E-Person: "Commpact Managers"

Commpact Manager: *A commpact manager is the digital process that keeps, manages, and interprets commpact objects that have been deposited with it. Commpact managers can be co-located with clients, servers (conventional access control), or trusted third parties (e.g., rights clearing houses). Every e-person has a default commpact manager.*

Commpact managers are essentially the network-centric realization of a conventional authorization reference monitor—using relationships as the unit by which to modularize this authorization module in a distributed environment. Commpact managers are servers specialized for security and access control; they are the entities that manage control information in the same way as we have servers managing content information. Commpact managers essentially act as a "personal relationship managers" for the e-persons whose relationships they maintain.

Commpacts can in principle reside at any commpact manager on the network. One and the same commpact can also reside at more than one site. Such replication would basically correspond to the real-world case where an agreement is distributed in copies to the agreement parties instead of being deposited with a third party. In particular, replication to the client-side is useful in case of "usage" rights; replication to a trusted third-party is useful in certain cases of lacking trust or of the need to access large amounts of data that cannot be moved elsewhere easily.

4.1.5 Reifying Standard Contract Templates: "Commpact Forms"

Commpact Form: *A commpact form is the basic "template" of a commpact. Commpact forms are much like a standard rental agreement in that they have been carefully designed by someone once, but then they are readily available as "stationery" to everyone else; general users can simply take such a form, customize it, fill in some parameters (such as the actual price offered, etc.), and then declare it an offer. Commpact forms are assumed to be designed by what we call a "commpact forms designer", and they are made accessible through "commpact forms providers."*

Commpacts are based on shareable (possibly standard) commpact forms. Forms are objects themselves. From every commpact, one can obtain its basic form. The notion of having standard templates of commpacts serves the purpose of keeping things simple for end-users and dealing with complexity at design time. At the same time, any qualified person can define a new form and make it available to others.

Standard forms make it possible to encapsulate possibly complex behavior at design time, and they provide a mechanism by which parties other than the ones that are directly involved in the negotiation of a specific relationship can contribute useful work. In other words, having a mechanism for forms defines a market for offering services that make it easier for interested parties to draft offers about certain types of relationships.

Encapsulation is also significant in terms of the third-party reputation that such bundles can acquire, and the reputation-based efficiencies that this introduces for end-users. For instance, general consumers usually find prepackaged choices more usable than micromanaging a larger number of individual decisions, some of which they might not care about in detail, others of which they might find themselves not even to be competent to decide about.

"Commpact Forms Designers": Developing Shared Commpact Forms

A forms designer is the original contributor of a commpact form. Such designers should have domain expertise combined with an ability to provide the corresponding implementation that is necessary together with various textual descriptions to make up a complete commpact form.

Defining a new form is analogous to coming up with a new standard rental agreement form, that is, a task that most people would generally not take on themselves; at best they would want to customize an existing form. While in principle anyone could define a new commpact form, we expect that this will be an infrequent case; new forms are likely to be provided by professionals at trusted proxies, such as home provider administrators, librarians, security officers, or an informal variety of the kinds of "local developers" that Nardi [250] finds are so useful in helping people use applications such as spreadsheets.

It is the forms designer's authority that guarantees the fact that a commpact's behavior corresponds to what its descriptions say, i.e., that the semantics of the textual descriptions and the semantics of the behavior of the associated implementation code are compatible. It is then up to the users of such a form to decide whether or not they trust this implementation, based on the reputation of the forms designer and other third-party information.

Forms designers do not necessarily need to be forms providers themselves; they do not even necessarily have to be online. Forms designers are therefore not explicitly reified (in any specific other form than being referenceable as an e-person). In particular, there can be a multiplicity of such forms designers in a way that lays open the different points of view one might have. For instance, one might have forms recommended by the Electronic Privacy Information Center, forms recommended by the Direct Marketing Association, etc.

Commpact forms can have different degrees of customizability. While some might limit customization to the option of crossing out a certain obligation or a certain right, others might allow people to insert different kinds of constraints and add more personal preferences. Designing a form involves providing an implementation of those basic methods of a commpact object that one wishes to override. Part of this is providing textual elements from which textual descriptions can be generated that describe the state of the commpact.

4.1.6 Making Available Commpact Forms: "Forms Providers"

Commpact Forms Provider: *A forms provider is the service that actually operates an online server carrying a collection of commpact forms that is searchable or browsable.*

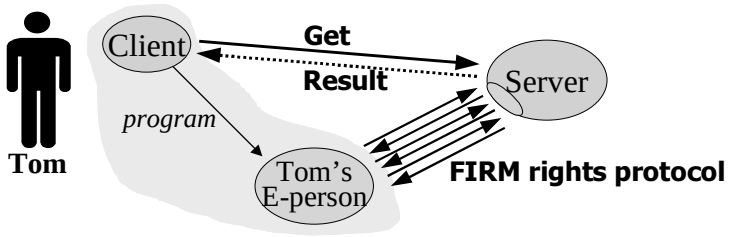
Forms providers are a subclass of commpact managers that specialize in commpacts in their 'form' state. Forms providers will often be the same authority as the designers, but they are not required to be so.

4.2 Transaction Model

FIRM defines a programmable infrastructure for negotiating new relationships and for authorizing actions based on existing relationships by exercising rights of a previously negotiated agreement. This protocol mirrors contract law practices.

As indicated in Figure 17, the bulk of the low-level transactions in FIRM are kept away from users by having a user's e-person agent execute and respond to FIRM protocol actions on a user's behalf. Users only specify the basic preferences that guide these actions, and the e-person takes care that any complexity remains invisible "under the hood" unless a certain case is not covered by any default rule. In RManage, for example, users can use an 'e-person control panel' to articulate basic preferences such as which offers an e-person should automatically accept or which obligations it should automatically fulfill. In this way, many actions can be dealt with automatically without the need for a user to deal with low-level issues.

FIGURE 17. Transactions in FIRM.



Note that while the client-server transactions can use conventional protocols (e.g. HTTP), the FIRM protocol can be based on more sophisticated mechanisms (e.g., a distributed object infrastructure such as IIOP/Corba [242]). FIRM itself is neutral to the type of protocol used for this purpose, although the current specification is described in terms of a distributed object environment.

The FIRM protocol has been designed with the following goals in mind:

- *Simple cases are simple in the FIRM protocol.* In particular, a typical specialization turns out to be essentially the same as standard HTTP authorization.
- *Complex cases are uniformly possible.* The main point is then that the FIRM protocol uniformly extends to cases that are not possible in existing protocols, such as negotiating new relationships. It accommodates sophisticated negotiation and con-

trol behavior, although the number of message exchanges will of course scale with the complexity of what one tries to accomplish.

In this section, we describe both of the transactional modes of FIRM: the negotiation mode and the performance mode. A more formal specification of FIRM can be found in the Appendix.

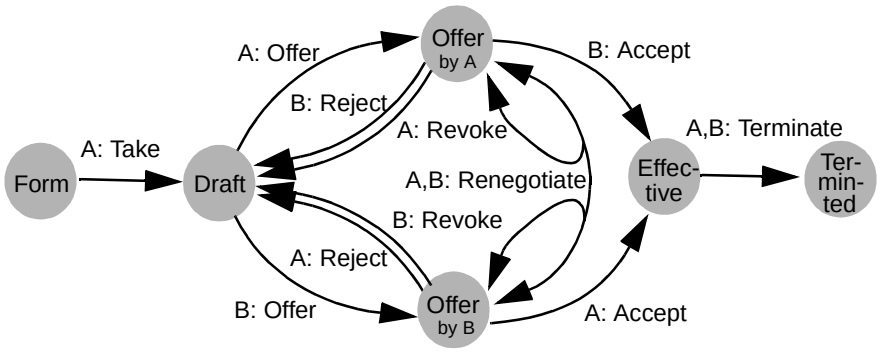
4.2.1 Negotiation Mode: Establishing Mutual Assent About an Agreement

Commpacts are agreed upon as a result of a negotiation according to a general, domain- and content-independent protocol, which is designed to reflect legal contract practices.

Negotiation States and Transitions

As we outlined in our description of the conceptual commpact model in Section 2.0, the basic actions are that of issuing an offer, negotiating it, and accepting or rejecting it, or revoking it by the party who issued it. Successfully formed, "effective" agreements can also be terminated and renegotiated—in which case a new offer takes the place of the previous offer, and a new negotiation is started. This is indicated in Figure 4 on page 17 which is repeated here for convenience as Figure 18. This figure shows a finite-state diagram that defines the overall process that can lead to a successful contract formation.

FIGURE 18. Negotiation: States and Transitions. (Repeated)



While Figure 18 indicates the basic transitions, at the action level, we would like to distinguish between the act of requesting a certain transition to take place and the act of declaring this transition to have taken place. This gives us the following set of protocol actions, two for each transition type:

`RequestDeclareItAnOffer`: For a given set of promises as part of a newly drafted commpact, request that it be declared an offer.
`DeclareItAnOffer`: As above, but actually declare it an offer now.
`RequestAcceptOffer`: Issue a request to accept a given offer.
`DeclareOfferAccepted`: Declare this offer to be actually accepted.
`RequestModifyOffer`: Issue a request to modify one or more attributes of a given offer.
`DeclareOfferModified`: Declare an offer to be modified in one or more attributes.
`RequestRevokeOffer`: Request an offer to be revoked if possible.
`DeclareOfferRevoked`: Declare the offer to be actually revoked.

`RequestRejectOffer:` Request the rejection of an offer.
`DeclareOfferRejected:` Declare the offer to be rejected now.
`RequestTerminateCommpact:` Request the termination of an effective commpact.
`DeclareCommpactTerminated:` Declare the commpact to be terminated.
`RequestRenegotiate:` For a given commpact, request a renegotation of it.
`DeclareAcceptRenegotiate:` Declare that a given commpact is under renegotiation.

There are a number of reasons why we introduce two different actions (one for requesting and one for declaring), including the fact that

- this accommodates the case in which different parties (e.g., another agreement party, an independent third party, etc.) execute the different parts of one type of action; and
- this allows the model to reflect situations in which actions have been requested but not yet succeeded.

As an example consider that we have successfully formed a commpact that one party would like to terminate. This party would then issue a `RequestTerminateCommpact` action. In many cases, this might automatically lead to a `DeclareCommpactTerminated` action. However, in some cases, the request action might require authorization from the other commpact parties—and the corresponding declare action would take place only after having obtained this authorization. In particular, note that there could be a time delay during which the state of the commpact is such that termination has been requested by one party, while the commpact still gathers information about whether this party has the right to terminate the agreement.

Another case to consider is that request actions can include other parties in the protocol in a structured way. For instance, there might be a "supervising" third party that certifies the negotiation process, as in many real-estate negotiations. Then the agreement parties would issue the request actions to this third party, which would then in turn issue the corresponding declare actions.

Often, shortcuts will be possible that avoid the need to have a request action separately from a corresponding declare action. In particular, if a party can successfully directly declare a commpact to be terminated, then there is no need to request this termination first—and wait for a corresponding declare action in the successful case. The request actions are only provided to cover a larger set of interactions, and to lead the path towards an implementation model where such actions can be executed asynchronously.

In the above protocol, we also incorporated actions for submitting and accepting requests to modify a given offer or agreement. This is to accommodate typical practices as well as transactional efficiencies, more than it necessarily reflects legal theory. If a merchant offers a good for a certain price, and an interested buyer asks for a price reduction and the merchant agrees, then the legal framework presumably would either interpret the change request as an "out-of-band communication" that leads the merchant to issue a new, adjusted offer, or it would posit that there was a counter-offer issued by the buyer that automatically cancels any previous offers between the two parties about this object (switching the offeror-offeree roles though). However, the latter is clearly a fairly inefficient mechanism transactionally for such a common practice. We therefore include additional protocol actions to submit and accept requests for changes in a given offer, and endorse the out-of-band communication interpretation.

"Race Conditions"

In a distributed environment of concurrent processes, a number of "race conditions" can happen that might lead to an arbitrary state if they are not dealt with in the protocol design. Such concurrency-related race conditions are situations where two parties independently initiate an action without knowledge that the other party also initiated an action and, depending on which action is given priority, two different states result. As an example, consider the following two situations that are classic cases in contract law:

- Crossing Paths Revocation-Acceptance: It might happen that the revocation request for an offer crosses paths with an acceptance request. The question needs to be resolved whether the acceptance takes precedence over the revocation (and the offeror is bound by his offer), or whether the revocation has priority—in which case the accepting party will have reacted to an offer that then turned out to be not an offer any more.
- *Crossing Paths Termination-Authorization:* Another case is that a commpact is requested to be terminated by one party (that has the right to terminate the agreement), and the other one just requested to be authorized for another action. We need to have a policy by which we can resolve the question whether or not an action can be still authorized under such circumstances.

In general, the policies that we assume in such cases are motivated by the way in which these issues are dealt with by the various contract law principles (cf. [1][2][3][14]). An additional complication arises in these cases, based on the fact that different legal systems deal differently with each of these cases. In the United States, the acceptance will generally be considered effective on dispatch, while the revocation is effective only on receipt (cf. [1]:par. 136). In other countries, it is the receipt that matters in each case.

We resolve the issue of such negotiation-related race conditions by defining the granularity of actions in a way that makes it possible to implement solutions in an unambiguous way. In particular, since we intentionally separate each action into a 'request' and a 'declare' part and since we are dealing with an environment in which each of these actions will take place in reasonably short time (a property that is not true for the action as a whole, though), we can adopt the following policies:

- The time of an action is the time at which the method call is received at the requested object. Note that this also makes sure that all actions are measured with the same "server time"—thus making it unnecessary to implement complicated synchronization measures for the times between different machines.
- The mapping onto the legal framework is that the completion of a request action is interpreted as the dispatch of a message, while the completion of a declare action is interpreted as the reception of a message.
- Request actions can block the performance of concurrent request actions if their types collide in the legal system that governs the contract formation, that is, for instance, in the United States, a 'request accept' action would block a 'request revoke' action until either a 'declare accept' action follows or the acceptance fails.

In other words, we use a standard blocking solution to implement policies for concurrency-related race conditions.

User Interface Affordances

Let us describe here the basic affordances for the various participants in a negotiation process: offerors, offerees, where each can possibly be supported by an agent.

Offeror

From the perspective of the offeror (e.g. a publisher), there are affordances to

- *draft an offer.* People have some form of an editor that they can use to fill in concrete numbers into boiler-plate agreements, customize them, etc.
- *declare an offer.* When the drafting process is finalized, users have a way of declaring it to be an offer now.
- *revoke an offer.* Outstanding offers can be called back by revoking them
- *terminate an agreement*. Effective agreements can be terminated. In addition, in case all promises were declared to be fulfilled, agreements are also declared to be terminated.

Each of these actions is supported by a corresponding method on commpact objects (cf. the next section). Offers include restrictions pertaining to how long they will be effective. Declaring an offer creates the liability of an outstanding offer. It can only be retracted by a revocation request. Any action is itself subject to authorization and might therefore not be feasible. For example, terminating a commpact is only possible if one has a corresponding termination right; revoking it requires a revocation right, etc.

Negotiation starts with the drafting of an offer. Offers are based on drafts and forms. Forms can be obtained from searchable collections operated by forms providers. Moreover, from any existing commpact (draft, offer, effective, etc.), one can obtain the base form that it was created with. Once we have a form, we can turn it into a draft and customize it in various forms. Associated constraints and promises can be modified. Once the offeror is satisfied with a draft, it can declared to be an offer, available to a certain set of e-persons.

In other words, for the purpose of creating an offer, there is a way for users to

- search for and select a commpact form from one of a number of forms providers, say, by some form of browsing in the simplest case,
- customize it (by modifying, adding, or deleting constraints, promises, object designators, etc.) and fill in defining parameters, such as the actual price to charge, etc.,
- declare it an offer.

Offeree

From the perspective of the offeree, we have the ability to

- *view a set of other people's offers.* For instance, one might want to view all the commpacts applicable to a certain object that one is interested in, or all the commpacts one can have with a certain e-person.
- *reject an offer.* This might be useful in the context of a larger negotiation process in which an offeror might release offers in some sequential manner, and only offer new ones after the outstanding ones were rejected.
- *counter an offer.* Counter offers are essentially just offers in response to other offers. As with conventional contract law practices, a counter offer is assumed to automatically cancel any outstanding previous offers.

- *accept one such offer and make the agreement effective.* This will lead to a check of the commpact's precedent conditions. Then, if these are fine, a new offer will be "cloned" from the previous one (if it is valid multiple times). Most importantly, a commpact of status 'effective' will be created at a designated commpact manager.
- *renegotiate an agreement*. Agreements can be renegotiated at the suggestion of one of the parties. This will spawn a separate negotiation dialogue.
- *terminate an effective agreement.*

Along the same lines, creating an offer is much like creating any other object in that there needs to be some commpact that authorizes this creation.

4.2.2 Performance Mode: Making Use of an Established Agreement

As we have described in Chapter 3, commpacts effectively realize a distributed authorization reference monitor. Given a user, a commpact, and an item or service that the user is interested in, it can be decided whether this action is acceptable (endorsed by a right) or whether it might be required (by an obligation).

Server

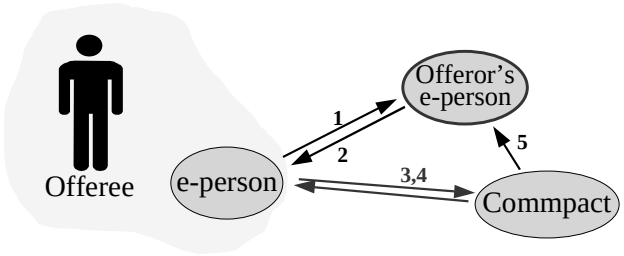
From the view of a server, FIRM primarily comes into play as an augmentation of the authorization module. Consider a request that is received at a FIRM-enabled server, e.g., an HTTP request to a Web server or some other request using some other protocol. One of two cases is possible:

- Either the request includes a handle to the actor's e-person, in which case we assume that the augmented authorization protocol of FIRM is to be used, or
- it does not, in which case we use any other default authorization, e.g., the conventional HTTP authorization.

Recall that after a "network login" (cf. Section 4.1.1), users are always identified by a (possibly public-key signed) token that gives a handle to their e-person. For example, if they use a Web browser as their client program, then the mechanism by which this handle can be communicated is by using HTTP cookies; these cookies are information that is sent along with requests to servers.

Assume we are now at the server, receiving a request that includes a handle to an e-person. The first step will be to authenticate this handle by checking it against the public key of the e-person's home provider. This will assure us that the request coming in from a certain network address is really associated with the e-person mentioned by the request—since we get a token that has a signature on the fact that this network address and the e-person are connected.

FIGURE 19. Negotiating a New Relationship.



1: Get all offers about item or service
2: List of pointers to offers
3: Accept chosen offer
4: Offer accepted
5: Offer accepted (optionally)

E-person

One of two cases is now possible again. Either the request already includes a reference to the commpact that is to be used, or it does not. If there is such a reference, then we proceed with this handle. If no reference was part of the request, we first need to ask the e-person about which commpact it wants to use—using the 'GetDefaultCommpact' method of e-persons that returns a reference to a commpact that is to be used for the given circumstances (cf. the Appendix). The e-person itself will act as an agent and use its default rules and preferences to determine whether it can uniquely identify an adequate commpact on behalf of its user. If it is able to do so, then it will return this one. Otherwise, the it will do one of two things:

- Either the e-person will indicate to the server that no default commpact can be determined. At this point, most servers will cause some form of a "Not Authorized"-message to be sent.
- Or the e-person will try to negotiate a new commpact that it can then use to access the service. See Figure 19 for this case. The e-person will ask the offeror's e-person for the offers that exist about the relevant item or service.
  - Then, in case this offer falls under the e-person's "auto-accept" preferences, the e-person will automatically accept it and then use it to perform the action.
  - If the offer does not qualify for automatic acceptance, then it will be added to the e-person's notifier. The user can then look at this offer and decide at liberty whether or not to accept it. Notifier objects might react in one of a number of ways to include the e-person's user into the feedback loop: by popping up dialog boxes if the user is currently present, by inserting it into an inbox if not present, etc.

Server

Once the server is pointed to the commpact to use, it will decide whether the commpact's interpreter is trusted and whether the commpact is a legitimate one. The latter will make sure that an actor cannot designate just any (bogus) commpact at an arbitrarily chosen site—and use it to authorize actions that the real rights holder would not have granted. A commpact is legitimate if it was really agreed upon by the parties that it claims to have been so; servers can ensure this by checking whether the promisor of the

commpact's promise that is to be exercised is indeed one of the offerors of the requested object at the server.[1]

Now let us assume that the server possesses validated handles to the actor's e-person and the preferred commpact. This is the basic input to an authorization decision. This decision consists quite simply then of

- getting hold of the right that we want to exercise,
- exercising this right.

Exercising the right without any exception corresponds to a positive authorization. If we get any kind of exception, then the action is either not authorized or its circumstances are more specifically qualified. For instance, consider a 'WaterMarkRequired' exception that would tell the server that it can send out the requested information, but it first needs to watermark it appropriately.

Note that the above appears complicated, but it really considers a lot of exceptions; in the normal case this reduces to a few transactions with an overhead that is comparable to current authorization schemes for the same types of authorizations.

In the case of a simple server-based relationship, the required rights object can be in the same address space as the server, and we should therefore hardly be able to recognize much difference in transaction cost between a conventional authorization and FIRM-enabled authorization. But with the additional infrastructure, we have a scheme that generalizes smoothly to more sophisticated authorization interactions, including those not possible with simpler schemes.

Client

As mentioned above, the conceptual assumption is that actions are performed with respect to the context of a designated commpact—which will then be used for authorizing this action. In other words, from the client's view, we need to have a way of communicating a reference to the commpact with respect to which we want to perform an action. In FIRM, this task is split up between the (human) user and its e-person—the e-person acts as an agent and tries to take over all those decisions that its user would have taken in all likelihood as well (based on user-determined preferences); it would then automatically pick an applicable default commpact, thus reducing the overall transaction costs of the contracting scheme. A possible user interface implementation might look like the following for a given action:

- If the actor is presently using the client interface and explicitly designates an "override commpact," use this one.
- Otherwise, check whether the e-person is able to identify an applicable default commpact. If yes, take it.
- Otherwise, enter negotiation: Ask the owner (or any other provider) of the requested object/service about which offers exist for it, etc.
- Let the e-person determine whether it can automatically accept one of the offers based on its preference rules. If yes, accept the offer and then use it for performing the action.

1. Note that one object can have more than one offeror, but adding a new offeror to an object's list of offerors will require a corresponding right to do so.

- Otherwise, add a message to the e-person's notifier with a selected list of offers as an attachment.

In Section 4.3, we will see an example of this behavior.

4.3 The User's View: Examples from the RManage Prototype

In this section, we describe how the kinds of affordances that FIRM enables appear at the user-conceptual level. We use the RManage implementation of FIRM as an demonstration example. Specifically, we will also introduce the RManage/DLITE viewer, a prototype relationship management interface targeted at expert users, including publishers, librarians, and other information/service providers, that was implemented as part of the Stanford Digital Libraries Project using the DLITE toolkit [249] and the Stanford Infobus infrastructure [253]. The Stanford Infobus is a CORBA-based distributed object infrastructure that provides higher-level information management service layers for managing items and collections, metadata, search, and payment. The RManage/DLITE viewer runs side-by-side with a conventional Web browser and provides augmented direct-manipulation affordances to make it easy to interact with FIRM objects—in a way that is integrated with the Web browser via HTML forms, etc. Note however that alternative interfaces could be implemented on top of the FIRM platform as well.

At first, we will look at the user interface affordances for general users. Then, we will describe the additional affordances that are available to those who offer new relationships.

4.3.1 User Interface Affordances for General Users

RManage provides for four major types of affordances for general users: Affordances for identifying oneself, for viewing and manilpulating the state of one's relationships, for controlling what to delegate to one's e-person, and for controlling the extent to which one wants to be informed about events in different relationships. We look at each of them in turn.

Identifying Oneself: Network Login

Identifying who a certain user is and authenticating this information is a necessary step that cannot be avoided, although it can be designed to incur less overhead than is the case in many current systems. In particular, we provide a network login facility that requires people to authenticate themselves only once. Once logged in, all subsequent, service-specific authentications that might be necessary are dealt with by the system, not the user. Note that while there exist network login mechanisms for limited domains, the Web currently does not have a standard mechanism for a network login. This makes it necessary to register again with every new service.

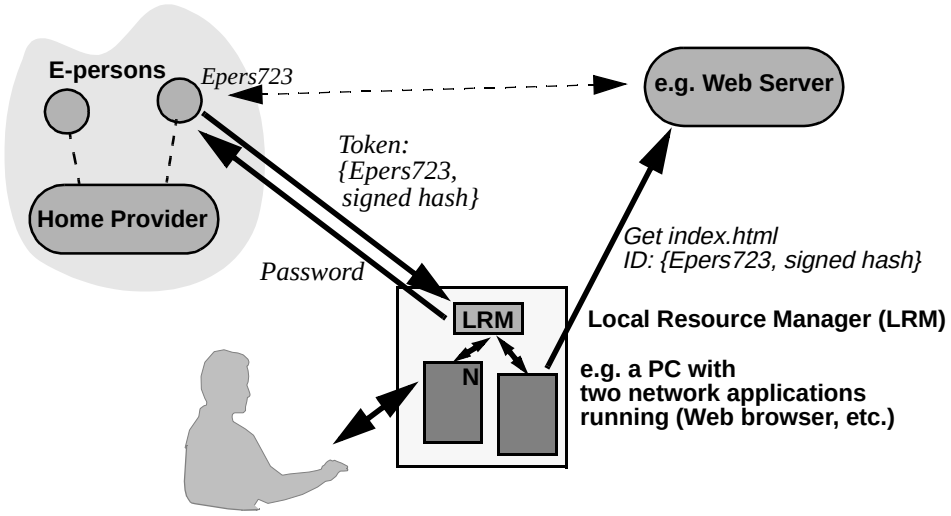
RManage implements a network login by initially authenticating a user's client application(s) with respect to the e-person of the user. These client applications can then include a handle to the e-person object when performing network requests, and any service has then a way of getting back to the user's e-person.

From a user's perspective, there exists a client program (e.g. a form in a Web browser) through which users can provide simple authentication information, such as a password. This password is used by the e-person to authenticate the identity of a user. From then on, all programs know who the user is and how to interact with any server on the net-

work. Note that in specific usage contexts, we can even further reduce the overhead that is created by the need to provide a password. For example, when running programs as part of a Unix account, then RManage can make use of the authentication required to log in to this account.

Under the hood, the following interactions would happen as part of such a typical network login. Let us consider the case of the user interacting with a networked PC, say, running two applications.

FIGURE 20. Network Login Interactions.



First, let us describe the notion of a "Local Resource Manager" (LRM). The LRM is an object that knows about the local resources that a user has around the physical location where he or she is using a specific computer. For example, the Local Resource Manager object will know about the fact that a certain printer is part of the same work environment as a certain display monitor. Note that the need for an LRM is created by the fact that client programs can essentially anywhere on the network. Two such applications might not even know that they appear on the same screen to one specific user. Note that LRMs are a necessary entity in a networked system based on distributed objects; they enable us to view a document and then print it locally for example—something which is not possible in systems without an LRM object, such as Marimba's Castanet. For further detail, see also [249].

Consider now the following login interactions, reading Figure 20 from the user clockwise around a circle. A user interacts with some client program and provides appropriate authentication information (e.g. a password). This client program will then contact the "Local Resource Manager" (LRM) to ask it to perform a network login. The LRM in turn will contact the user's designated e-person, and obtain an authentication token in exchange for a correct password. This token is a handle to the e-person object, signed by the home provider's private public-key-cryptography key. The LRM will now initialize any of the user's client-side programs (e.g. a Web browser) with this token such that this application can send it along with every request. This allows then any of the subsequently contacted services (e.g. Web servers) to talk back to the user's e-person and automatically negotiate further access conditions (dashed horizontal arrow). Note that,

for the last step, application-specific mechanisms need to be exploited. For example, to have a Web browser send along an identification token with every request, the LRM can use the HTTP cookie mechanism that is now available in most Web browsers.

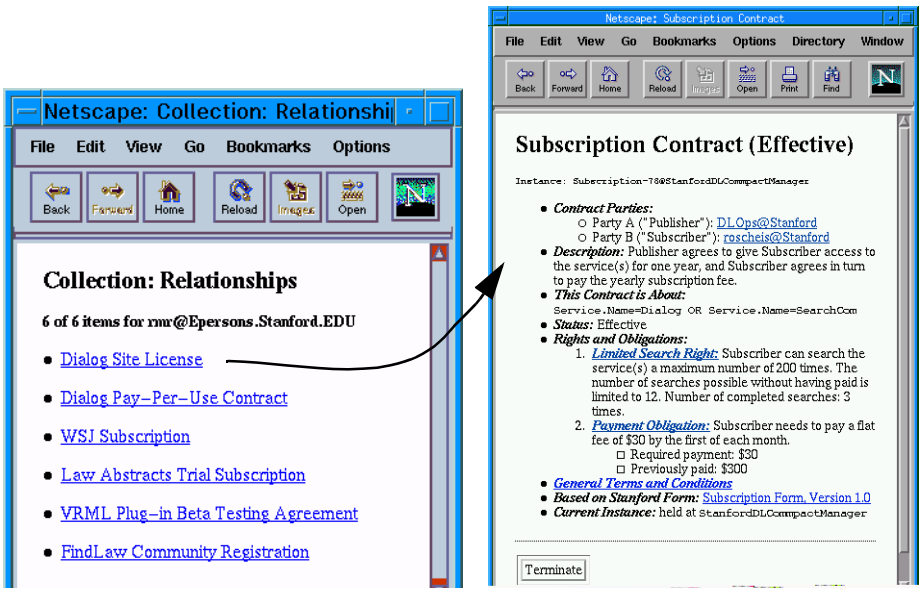
The above scheme generalizes readily to the use of multiple e-persons. In this case, we will generally only require users to authenticate themselves with respect to one e-person, which will then perform further authentications to other e-persons if applicable.

Viewing and Manipulating One's Relationships

RManage provides users with a uniform interface to the relationships that they have with the various providers of FIRM-compatible network services. Figure 21 is a sample view that shows how users can list their relationships, and how they can initiate associated actions such as payment transfers in the Web-based RManage client in their notifier.

For each of the relationships ("digital contracts"), a page that provides a structured description of the relationship's current state can be obtained by following the link in the listing. This page will then provide a generic set of affordances that are common to all relationships—such as the ability to terminate—and a set of type-specific affordances, such as renewal and payment options.

FIGURE 21. Relationship View in RManage.



Controlling What to Delegate to an Agent: E-Person Preferences

By using an e-person, a software agent can take over replying to the bulk of those kinds of requests for which it is clear how the user that the e-person represents would have dealt with. The exact degree to which such requests will be automatically answered is a user choice that can range from "fully automatic" to "always ask me explicitly." Requests sent to an e-person might include requests to accept a certain offer. Obviously, nobody would want his or her e-person to automatically accept every such request, since

this might result in unwanted liabilities. Therefore, more reasonable preferences for the e-person would typically include the type of offer, the size of the liabilities created, etc.

RManage implements two simple forms of e-person preferences to reduce the social transactions that are involved in negotiating new agreements: Auto-Accept and Auto-Fulfill:

- *Auto-Accept:* a way of designating which kinds of offers can be automatically accepted by a user's e-person without the user being explicitly. Possible parameters include the type of the agreement (e.g., "accept all agreements that do not lead to any obligation", "accept all agreements that only give out my ZIP code"), the place where they are enforced ("accept all user profiling agreements managed at Nielsen's site").
- *Auto-Fulfill:* a way of indicating what kinds of obligations to fulfill automatically. Parameters for this might include the type of obligation (e.g., "fulfill all payment obligations just before they are due"), specific parameters of these (e.g., "fulfill all payment obligations that are less than $5"), etc.

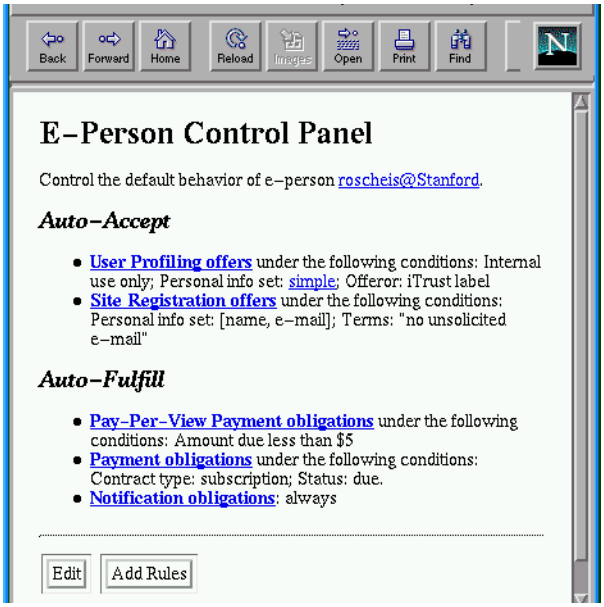
Note that this allows, for instance, for new agreements to be negotiated just by following a link in a Web browser, without any explicit user involvement.

The screenshot in Figure 22 shows an example page describing the preferences for the default behavior of e-person 'roscheis@Stanford'. As with all of the browser screenshots in this section, note that the HTML pages that they display are textual descriptions that were generated from FIRM objects directly (in MIME content type text/html), based on the structured descriptions that each of these objects contains. For example, a FIRM payment obligation object might have a 'price' object property that can be requested by client programs using the FIRM protocol. At the same time, there exists a URL that when requested by browsers will make the distributed object return a textual description that shows the value of this attribute. The design of these descriptions is partly embedded in the FIRM toolkit that is part of the RManage implementation, and partly due to the specific way in which a forms designer chooses to implement a new commpact object.

Figure 22 shows one such page, generated directly by the corresponding e-person object. We see that the e-person would automatically accept all those agreements offered to it that are based on one of two forms: it accepts use profiling agreements if the requested attributes are for internal use only and if it is operated by a trusted site. Site registrations are also automatically accepted if only name and e-mail address are requested for this purpose, and if the e-mail address is not used for unsolicited messages. Furthermore, pay-per-view interactions are limited to a certain maximum price unless they come from a subscription that was previously set up.

Note that there is full end-to-end integration. If a user presses one of the "Pay Now" buttons in Figure 22 in order to initiate the fulfillment of a payment obligation, then a structured set of actions is triggered both at the service provider's side (such as a database update) as well as at the user's side (such as an addition of an appropriate item to the user's check-book application).

FIGURE 22. E-person Preferences.



The correspondence between textual descriptions and underlying actions (i.e., whether clicking on 'accept' really leads to an accept preference) is matter of the implementation of the object-request interface of the relevant FIRM object. In the case of commpacts, this implementation is provided by forms designers, who would then also stand in for "truth in advertising." For e-person objects, home providers will want to make sure that they use a faithful implementation; FIRM does not specify any mechanism for getting such an implementation. In fact, FIRM also abstains from specifying the user-to-e-person protocol since there are no interoperability requirements for this "private" interface.

Controlling Access to One's Attention: Notifier

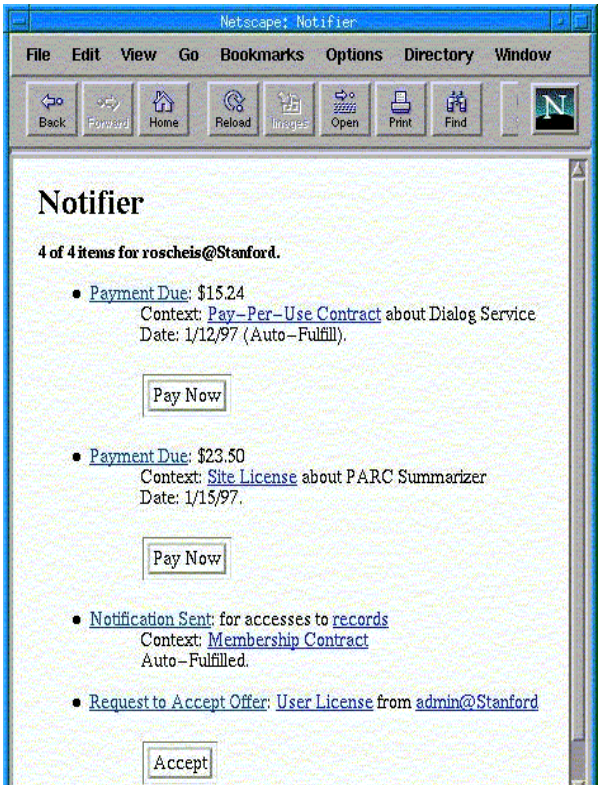
Events from relationships are brought to a user's attention in a uniform way in the notifier structure. The notifier is an affordance by which a user can catch up with 'what's new' for a certain e-person. This transparently includes both information push as well as information pull events; the notifier implementation takes care of the specific access method and structures the results in a way that mirrors the user's preferred attention structure. The notifier also allows people to initiate certain actions, such as fulfilling payment obligations, and accepting offers.

In the current RManage implementation, the notifier basically only acts as an inbox that allows us to take some modality out of interaction design—since we have obligations as first-class objects, we can just add them to a user's notifier and rely on the fact that they are enforced to the extent that they will not go away unless they are fulfilled. In Grassroots [279], we have explored the notion of a notifier more extensively with respect to information organization and communication.

Figure 23 shows a sample view of the notifier of e-person 'roscheis@Stanford' queried at a certain point in time. We see that there are various payment obligations due (the hyperlinks lead to more detailed information about them and their context); affordances

are provided for directly initiating a payment transfer to fulfill them. Moreover, we are informed about the fact that a notification obligation has been automatically fulfilled and that an approval request is pending.

FIGURE 23. Notifier: Uniform View on Events from Different Relationships.



Declaring Overrides for Special Cases

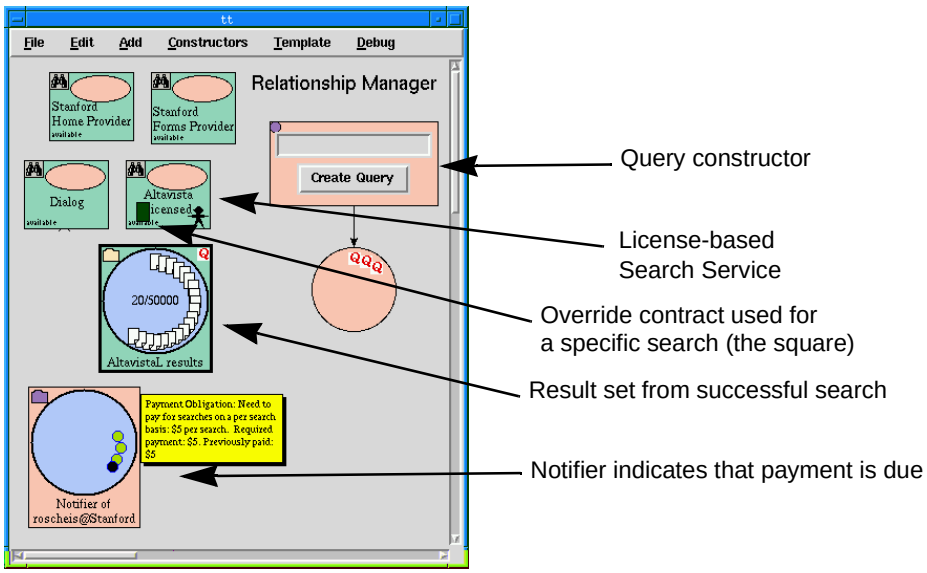
Recall that when performing an action, users have the choice to designate a commpact with respect to which they want to conduct the action. This commpact then acts as the unit for authorizing a certain action. If the e-person's preference rules clearly indicate which agreement is applicable for a certain action, then RManage-enabled clients will automatically include a reference to this default agreement in the request.

For example, consider that we have a subscription agreement with a certain online newspaper, and that we have chosen this agreement to be the default agreement to use when accessing the newspaper. Then, whenever we access material of the newspaper, the user's e-person will designate this agreement to the publisher. In other words, in the normal case, all the required rights-management interactions can happen transparently to the user.

Now consider that for some reason we want to use some other relationship to access to same material. For example, consider that we also have a site license to the newspaper. One way to deal with this is to declare the site license to be the default agreement for accesses to this newspaper. Another affordance that RManage provides is that of a temporary "override" to the default agreement. This provides a more light-weight way of

using another agreement for one or a few actions and then going back to the default agreement. In RManage/DLITE, we have experimented with various direct-manipulation ways of indicating such overrides as a feature for expert users. Cf. Figure 24 for an example of how this is prototyped in the RManage/DLITE interface.

FIGURE 24. Declaring Overrides in RManage/DLITE: Dropping a Contract Icon on a Search Service.



Once accepted, an agreement enables users to use the services covered in the agreement. The promises in an agreement implement the specific fee structure of a for-pay service. For example, in a sample search license for the Dialog service as part of the RManage implementation, we provide a fairly sophisticated search right that allows for a limited number of searches before the payment obligation has been fulfilled. The payment obligation on the other hand implements the fact that searches in this agreement are charged for on a per-search basis, etc.; it then uses any of a number of native payment protocols to allow this obligation to be actually fulfilled if this is what a users wishes to do.

In the direct-manipulation interface in Figure 24 (RManage/DLITE), such obligations, as well as agreements themselves, can be directly manipulated (transferred to other people, etc.). Payment obligations are some of the items that typically will show up on a user's notifier. In some cases, these notifier messages will only indicate that a certain payment is due at a certain time. In other cases, especially if used as part of a pay-per-view contract, they will often be fulfilled automatically and the notifier might only contain a message that summarizes the payments executed.

Note that RManage independently manages control and content objects and sets them in relation in a way that enables different control behaviors to become effective for one and the same object.

4.3.2 User Interface Affordances for Offerors

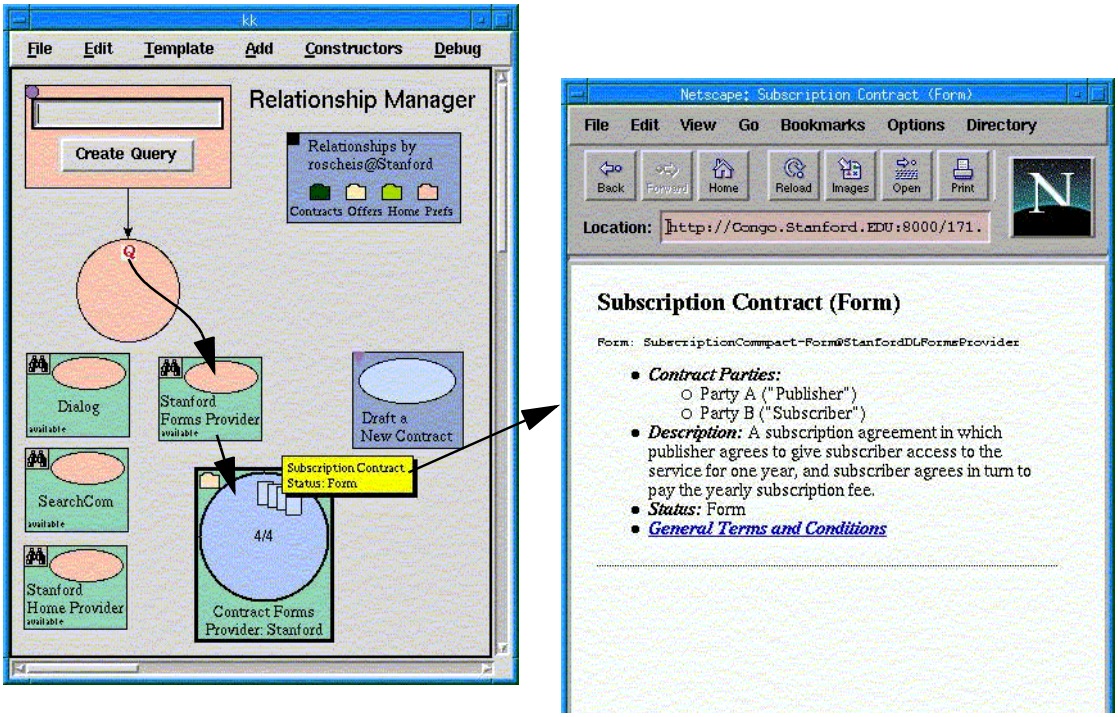
In this section, we describe the user interface affordances that RManage provides for those who offer information or services to others. In terms of our rights-management framework, offering a service to others essentially amounts to creating an offer that oth-

ers might then accept. FIRM was designed to make it easy to create new offers by defining an infrastructure for the sharing of customizable contract forms. The basic process of creating an offer consists primarily of selecting a contract form, customizing it, and declaring this draft an offer.

Obtaining a Useful Contract Form

Forms can be obtained in one of a number of ways, including browsing and searching. Figure 25 illustrates the offer drafting process as implemented in the RManage relationship manager application. The left side shows the RManage/DLITE direct-manipulation interface. The right side shows an HTML description of a contract form. In RManage, a user creates a query for ‘subscription’ (upper left) and employs the ‘Stanford Forms Provider’ service (middle) to search for digital contract forms that might be useful in drafting a subscription offer. A result collection is returned with four different commpact forms. Users can inspect each form and then take one and use it to draft an offer—by customizing it and by filling in form-specific parameters.

FIGURE 25. Using Commpact Forms to Make it Easy to Offer New Relationships in RManage/DLITE, a Java/CORBA-based direct-manipulation interface.

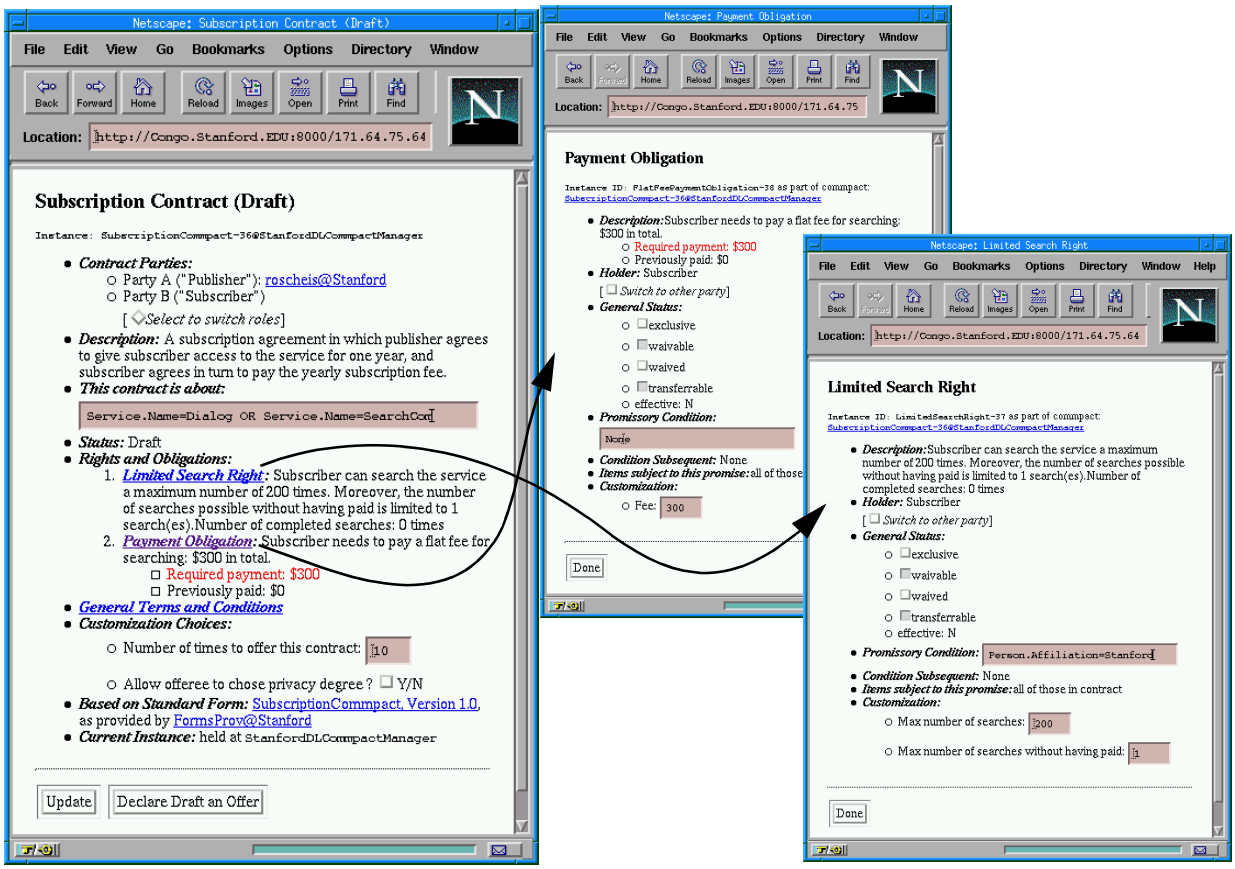


Drafting an Offer by Customizing Contract Forms

Once we have decided on a form and put it onto the drafting workbench, we can fill in the defining parameters and choose the various options that the form provides. For example, Figure 26 shows on the left the top-level choices for a sample subscription agreement. These include simple fields for issues such as in which role we want to offer the contract, how often to offer it, and a constraint about what the offer is. For each of

the two promises in this agreement, we then have further fill in options. For example, in the specific search right that this contract form incorporated we can set the number of searches that are allowed without having paid first. Moreover, we constrain the use of this right to people affiliated with Stanford University by adding a corresponding constraint. In the payment obligation, we can fill-in the subscription fee. In every case, there is a range of generic attributes such as whether a promise is waivable, etc.

FIGURE 26. Customizing and Setting Parameters in a Contract Draft.



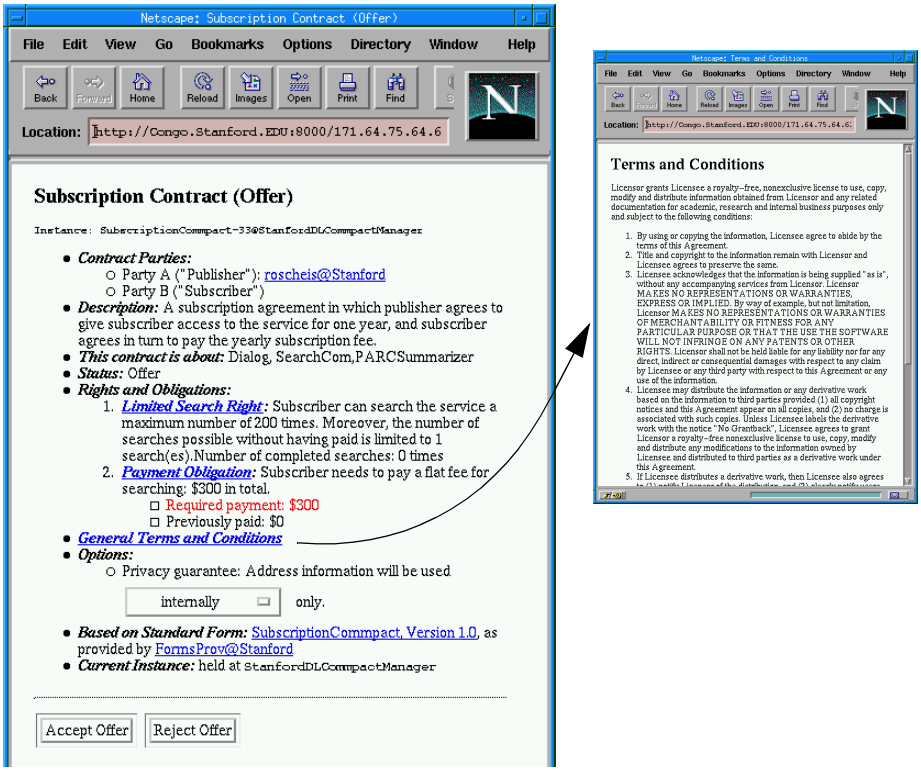
Values for attributes of persons can either be provided by the home provider or by third parties. For example, certain home providers might record the student affiliation of a person with their e-person structures; then, if trusted, these values can be used to obtain this attribute for a person. Otherwise, the home provider might only provide an identifying handle for the e-person (such as its full name), with which a commpact can then obtain the attribute from some third party such as the university itself. The exact way in which this is done is a decision of the forms designer.

Declaring a Draft an Offer

Finally, we can turn the contract draft into an effective offer that other users or agents can then examine and possibly accept. See Figure 27 for a generic presentation of an offer to users.

Note that in order for A to give access to B about I, in a contracts-based approach, A will create an offer about I that is limited to B, and B can then accept it if he wants to do so. Note that in a more property-based model we could have simply “attached” the access right for B to I and then tell B about it in some out-of-band way (in the contracts approach, we first have to assure mutual assent).

FIGURE 27. Sample Contract Offer: FIRM provides a way of generating structured contract descriptions.



4.4 Object Interactions: Sample Transaction Scenarios

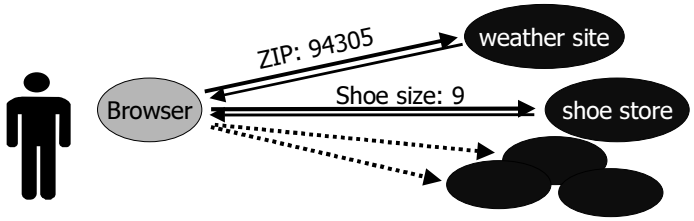
In this section, we examine a sample transaction scenario that demonstrates how digital contracts make it possible for new relationships to be rapidly negotiated and applied. The extended example that we consider here is taken from the privacy domain, and it shows how a contract approach can be extended to domains that have previously been amenable to a property approach only. We will also consider various aspects of a typical subscription contract in this section.

4.4.1 Example: Contracting for Privacy

When visiting a shoe store to buy a pair of shoes, few people would object to giving out their shoe size—since this is likely to make the shoe shopping experience more pleasant. The question is how the analogue would work in the online domain: how can users determine the nature of their relationship with the provider of a certain server, agree on boundary conditions, and then make available personal information in a controlled

way—the shoe size for the shoe store, the ZIP code for the weather site (to be able to get the local weather right away), basic demographic information for advertising-supported sites (to minimize irrelevant ads), etc. In particular, the question is how can this be accomplished without imposing any unnecessary usability overhead. Cf. Figure 28.

FIGURE 28. Online Privacy: RManage uses FIRM to allow users to reveal personal information in a controlled way.

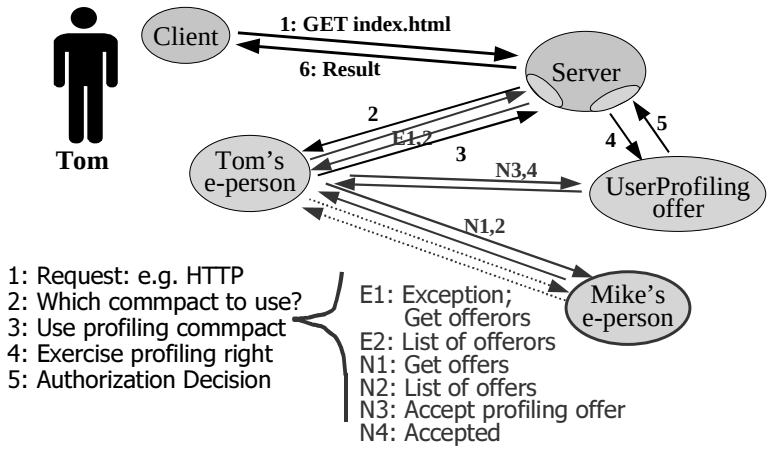


The user of a browser can set up his e-person to automatically accept user profiling contract offers of a certain type—those that are about a simple set of personal information and that have been certified by a certain labeling service. In the user's view, a first-time visit to a FIRM-enabled weather server will then present the user directly with the local weather—since the server could use FIRM to obtain the requester's ZIP code in a controlled and consensual way. However, while accessing the FIRM-enabled site appears to users to be as simple as browsing any other (uncontrolled) Web site, a whole range of FIRM transactions are happening "under the hood": A user profiling relationship is negotiated based on the user's default preferences, and personal information is used for agreed-upon purposes.

Figure 29 indicates some of the transactions that FIRM defines for an authorization that triggers the negotiation of a new relationship. Consider in particular the protocol requests labelled E1, E2, and N1-N4. Note that this is the most comprehensive case that tries to accommodate complex circumstances (no pre-existing relationship, no network caching, possibly multiple offerors, etc.). In particular, these transactions include:

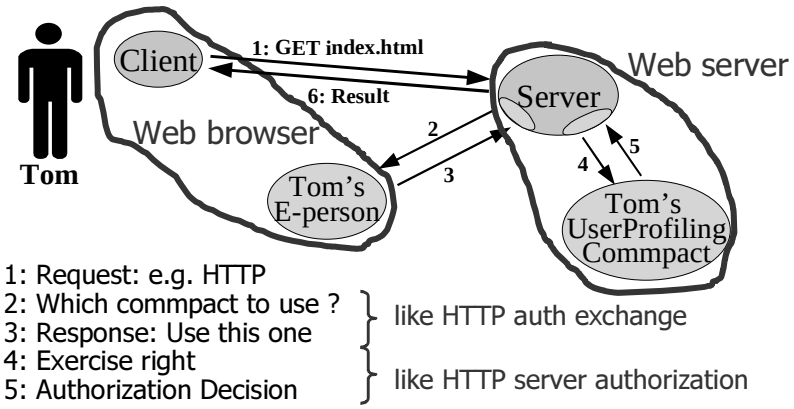
1. Requesting a document from the server, using some document access protocol (e.g. HTTP). The FIRM-enabled server recognizes that the user has an e-person, and therefore uses the augmented authorization module to negotiate access conditions.
2. The server asks the requester's e-person for the commpact with respect to which this access is supposed to take place.
3. The e-person returns a pointer to a commpact that it would like to use as the "authorization decision facility" for the requested action. It could also return an authorization token that it might have cached from a previous authorization.
4. Receiving the pointer to the commpact, the server will then ask this commpact for authorization.
5. Once it obtained authorization (and validated it), the server can perform the requested action.

FIGURE 29. Transactions Under the Hood: In one of the more comprehensive cases, a new relationship is established: Tom accepts an offer from Mike, one of the offerors.



The above holds for the case in which there was already a relationship established. Note that this protocol effectively mirrors other typical authorization protocols. For example, in one of the typical cases where a user's e-person and the browser are realized in the same address space, and the commpact and the server are co-located as well, then we essentially obtain the conventional authorization scheme of the HTTP protocol as a specialization of this special case. In other words, the simple cases reduce to familiar protocols while the more complex cases can be dealt with uniformly. Cf. Figure 30.

FIGURE 30. Special Case of Transactions in Case Objects are Allocated in Specfic Form.



If we have not yet had a relationship established, then the following negotiation protocol will be triggered (see the additional actions in Figure 29):

E1. The e-person asks the server to find out the identity of the offeror(s) for the relevant objects.

E2. The server designates relevant offerors.

N1. It selects an offeror (by the name of Mike in Figure 29) and asks the offeror's e-person about any relevant offers.

N2. The e-person receives pointers to relevant offers.

N3. The e-person inspects an offer, and based on its default rules, it can automatically accept it, and then use the accepted offer to authorize the access that required the negotiation of a new relationship.

Note that in many cases a server will of course be able to short-cut the above protocol, for example, by directly providing a default offer for a certain object, or by having information cached from previous interactions.

4.4.2 Example: Subscription Contract

As a second example, let us consider a typical subscription contract with its various terms and conditions. Consider a one-year subscription to an online newspaper that is available on the Web to subscribers who pay in regular intervals for the subscription.

Coverage

The fact that this contract is about publication items that for instance have been issued throughout the calendar year of 1997 is just a constraint as part of the FIRM contract that specifies that the publication date of the newspaper (an 'item') is in this year:

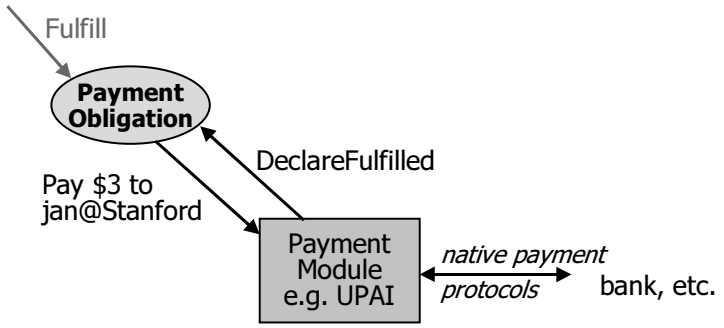
AboutItems Constraint: `Item.PublicationDate.Year=1997`

Note that this would allow us to access the issues of the newspaper from this year even after the year has ended or the contract was completed.

Payment

As an example that demonstrates how fulfillment processing is tied into the FIRM rights management service layer, let us now see how payment is dealt with in RManage in the form of one of various kinds of payment obligations. For FIRM, payment primarily appears in the form of payment obligations. FIRM keeps track of the status of payment obligations and provides affordances for fulfilling them, but the actual way in which they are fulfilled is dealt with by other service layers. The RManage implementation uses the UPAI [251] payment application interface. Cf. Figure 31.

FIGURE 31. Payment Interactions in FIRM.



As with any other RManage promise, payment obligations are first-class objects with a generic set of operations applicable to them. In the case of payment obligations, these

operations amount to initiating a payment transaction (fulfilling the obligation), declaring it paid, etc. The exact way in which payments are executed is a matter of specific payment protocols and institutions; it is not part of the FIRM specification itself.

While the RManage prototype implementation leverages the UPAI payment application interface [6], any other implementation can make a different choice here as long as the basic FIRM callbacks are used. UPAI provides an abstraction layer to integrate native payment protocols from a variety of providers such as First Virtual, DigiCash, VISA, etc. that allows RManage to disintermediate the various parties that play a role in payment transactions and provide a uniform interface to payment. Note that by leveraging FIRM's structures, RManage's interface provides a unified definition of what it means to assent to a contract, to fulfill an obligation, etc.

Discounts and Other Contract Options

Let us now consider the case in which a customer would like to buy a subscription from an online bookstore, but, as an ACM member, she would like to obtain a 10% discount. From the user's view, no breakdown should need to occur in order to deal with the discount; the bill in the end would just contain the appropriate deduction if the bookstore is FIRM-enabled.

To understand what is happening "under the hood," we need to clarify the contractual structure; once we have decided how to understand the situation, then there is a straightforward mapping into the RManage objects. Note that while FIRM makes control programmable and facilitates the computational realization of legal structures, it does not help people obain an understanding of the legal structure of a certain situation.

One way to conceptualize a discount is the following (the designer needs to be clear about this; not the user): The basic promises in the subscription contract are those of an obligation to deliver the publication (held by the bookstore) and an obligation to pay (held by the customer). To accommodate a discount, we assume that there is one and the same contract, but in addition we have a discount right. This discount right has a promissory condition that requires its holder to be a member of the ACM. When exercised, it simply reduces the amount to be paid (an attribute of the payment obligation object) by 10%. Of course, users or their e-persons need to remember to exercise this right in order to take advantage of the reduction. This is what one's e-person will generally do if it is set up to do so. Then other kinds of reductions can simply be reflected by adding corresponding discount rights to an obligation. Note however that exercising a discount right might of course also have other ramifications. For example, it might imply a waiver of other rights—for instance, it might waive the use of other discount rights, or it might void a right to return the good, etc.

In other words, FIRM forces us to be clear about the interpretation of a certain transaction (e.g., questions such as whether something is an advertisement or whether it is an offer at the same time). But it does not help us in finding this conceptualization; this is an issue that needs to be dealt with prior to using the FIRM framework. Once it has become clear how we want to understand a certain relationship, then FIRM provides a straightforward mapping of the (contract law) entities into computational objects.

Terms and Conditions with Arbitrary Predicates

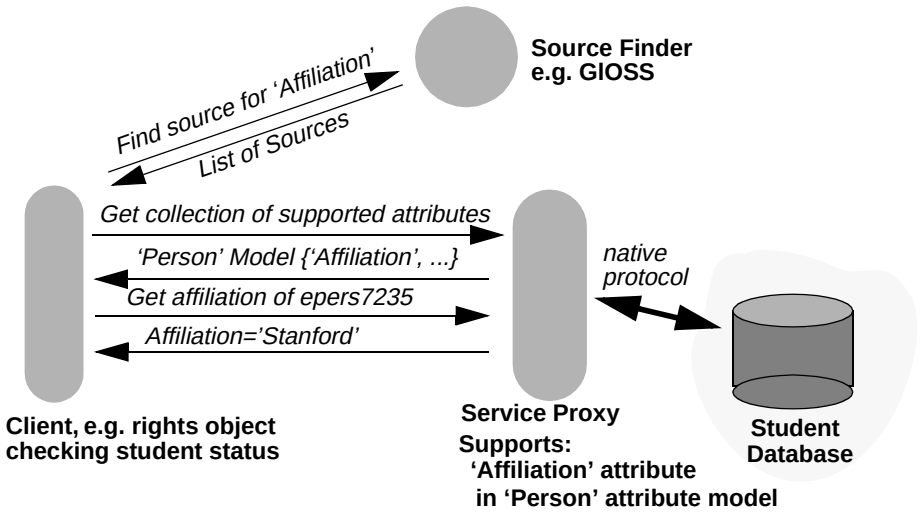
A vital part of a rights-management systems is to be able to articulate and enforce such constraints as "accessible to US citizens only," "minimum age of 12 required," or "academically priced–for currently enrolled students only."

FIRM provides for constraint structures as part of its rights objects that allow for policies to be formulated based on arbitrary predicates. However, FIRM does not specify any (idiosyncratic) constraint language for articulating such constraints. The way in which constraints are provided to FIRM objects is left as a choice to the specific user interface realized by individual commpact implementations. Furthermore, FIRM itself does not say anything about the internal representation or the syntax of such constraints; it only specifies a basic request interface that constraint objects obey and that can be used to determine whether a constraint is satisfied, etc. In RManage, we use a simple Boolean constraint language that users can use to type in constraints in the various dialogue boxes of commpact forms.

Predicates can rely in FIRM on any of the attributes that have been defined for a certain object by a certain attribute service (proxy). In particular, attributes for constraints can come from several sources. For example, a home provider could provide citizenship information while the student status of a specific person might be best obtained from a proxy operated by the university. In other words, implementations using FIRM will have to decide which sources can be trusted and which ones will not be trusted.

For more information about the kind of metadata architecture that FIRM requires, see the Appendix and [252]. In particular, FIRM leverages the online availability of services on the network and does not use cryptographic credentials where other systems might use them easily: For instance, to certify the student status of a person, a service can just directly send a query to an online database—rather than introducing cryptographic credentials that can then be "presented" by users to a service.

FIGURE 32. Certification: Example.



Consider for instance a subscription contract where the price is specially reduced for students. In other words, the condition precedent of the contract object is a constraint

that requires the purchaser to currently be a student.[1] One specific way to articulate this in RManage/FIRM would just be the following:

```
Condition Precedent: Person.Affiliation='Stanford University'
```

Such a constraint would rely on there being an attribute proxy such as the home provider that knows about a person's affiliation. The fact that affiliation is an attribute of the Person attribute model would be a property that can be read from the representation of the attribute model; it is a fact that can then be conveyed explicitly in the user interface that allows offerors to provide such a constraint (for instance, via an option box).

A more complicated attribute proxy might even provide a more general attribute such as StudentStatus (e.g., by relying on further requests to backend services); then, we could have a more general constraint such as

```
Promissory Condition: Person.StudentStatus='Y'
```

As we explain in more detail in the Appendix, FIRM itself does not specific any attribute sets. It uses a metadata architecture with first-class attribute models as a mechanism to keep this dimension outside the type system of FIRM. In other words, FIRM can be used as long as commpact forms designers and services use attributes from shared attribute models.

4.5 Related Work

FIRM relates to much recent work done to address the issue of security and rights management on the Internet. First of all, there is clearly a large amount of work in cryptography and classical computer security on specific mechanisms by which the validity of certain assertions can be securely determined. [171][164][163] Our perspective on this work is that we will use the best-of-a-kind products from this area, but the basic mechanisms for managing control information that are our focus here address an orthogonal layer of issues. In this section, we describe some of the key differences of our approach to systems that are roughly in the same design space.

The SDSI system [147] provides an integrated solution for distributed security that is based on simple, new representations for each of the different components that are necessary. One of the main differences to FIRM is that SDSI does not clearly separate protocol and architecture from specific implementation and transmission choices that can be made by different implementors. For example, SDSI defines a new object request interface (e-mail/HTTP), a new object transmission standard (the S-expressions), a new constraint language (the ANDs and ORs defined there), a new naming scheme, etc. Given the multiplicity and heterogeneity of developers that need to adopt such a system in order to make it work, it seems unlikely that it will be easy to get people to adopt all of these new formats, especially in light of the fact that there already exist multiple legacy solutions for each of these problems. Note that FIRM takes a somewhat different design approach in that it deliberately abstains from defining, for instance, a new constraint language; it just defines rights-related object interfaces using standard languages and components, by which different parts of the system can be glued together. Furthermore, SDSI does not define a user-conceptual model. In addition, it does not clearly

1. Alternatively, we could also install the same constraint as a promissory condition of the right to a discount (with somewhat different ramifications).

separate the distinct layers of articulating assertions and their expression using specific cryptographic mechanisms. This ties the architecture to a specific use of cryptographic methods.

The work on rights management done at Xerox (Stefik [97][99]) contains (at least) two major components: a language for describing rights (the "Digital Property Rights Language" (DPRL) [99]); and a specific architecture to assure the secure interpretation (enforcement) of this rights language. The rights language describes rights in a certain domain; it does not intend to provide a programmable framework, within which several such rights languages can be expressed. The vocabulary of the rights language is very rich (or, "complex") compared to typical other rights languages. Its main focus is on representing the kinds of (property-based) rights that are subject of the Copyright Act, such as restrictions on copying, fair use rights for "backup" copies, etc. There is also some amount of contracting/licensing implied, albeit not represented explicitly in a direct form: A "ticket"-based conceptualization is used to deal with licensing relationships rather than a full-fledged, direct contract-based approach. The architecture that assures secure interpretation is based on a Trusted Computing Base (TCB) that extends beyond what other secure content container architectures would typically assume.

IBM's cryptolopes [144] are an instance of a secure content container architecture. Along the same lines, InterTrust [145] has been developing a proprietary architecture for secure content containers that generalizes IBM's approach in that it does not require any built-in architectural centralities. InterTrust's architecture for managing control information accommodates first-class control objects, called 'control sets', although the emphasis is on client-based control (using their secure content container technology) rather than on network-centric control as in this thesis. In our view, the difference between InterTrust's client-centric and FIRM's network-centric architecture largely boils down to the question of the extent to which one believes that users will want to be micromanaged in client-side actions.

FIRM is designed to be orthogonal to any content container technologies. In fact, it complements them, and it can use best-of-a-kind container technologies for enforcing the terms and conditions expressed in commpacts if content objects need to be tightly kept under control after release. Effectively, commpacts correspond to an access-control decision facility, while secure content containers provide an access-control enforcement facility.

InterTrust and FIRM share an overall emphasis on having a programmable platform. For most practical purposes, FIRM is far more light-weight, however, especially because it clearly decouples enforcement and authorization issues, allowing application developers to leverage enforcement mechanisms that work well for applications in a certain domain. InterTrust claims a number of patents on its technology, none of which seem to be related to content container technology or software-based rights management, though. Details of the InterTrust design are not published.

EDI (Electronic Data Interchange) is a widely deployed message-based infrastructure for electronic contracting that has the kinds of mechanisms for negotiating agreements and defining new forms of agreements that are not very well represented in conventional digital rights-management systems and that fall into the layer we are focusing on. In particular, the notion of a standard template is one of the key insights from the EDI world that we incorporate into our architecture. Creating a new commpact form effec-

tively corresponds to the case in EDI of creating a new message interaction type. Note that in EDI this requires submitting a “New UNSM Request” to the appropriate “RT Secretariat,” that is, the “rapporteur team” appointed for a certain geographical area, which will then coordinate with the appropriate UN ECE office whether the proposed type is sufficiently different from previous ones and warrants a new type. In the commpact model, we carry over the assumption from EDI that new commpact forms are generally not set up by end-users. This thesis does not address the question of the extent to which an institutional infrastructure is required to support the naming and proliferation of the various standard forms.

The World-Wide Web Consortium (W3C) has recently started a number of efforts along the lines of privacy (P3P [219]) and personal context management (“Web passports” and OPS [220]). Both embody a number of key concepts that have also been part of FIRM. The W3C proposals are more specifically targeted at providing fixes for certain immediate problems, with the possibility of incremental improvements—rather than starting out with a more general design that deals with a wider range of usages.

Mondex together with an industry consortium of companies has pioneered a higher-level purchasing protocol, the Open Trading Protocol [227]. This protocol defines requirements for consumer purchasing contracts that can be seen as a specific instance of the kind of contract, that are possible in FIRM. At this point, there does not exist a technical protocol/architecture specification though; only a requirements specification exists, and there is no evidence that this work is actively being pursued. The current requirements specification deals with a subset of the issues that are addressed by FIRM.

The design of the TIHI system [184][185][186] addresses security in collaborative settings by providing a gateway, owned by the enterprise security officer, that uses a set of policies to mediate queries and responses. The policies of the gateway include release rules that make it possible to enforce content-based access control. TIHI is a practical implementation of an architecture that takes into account the availability of a (human) security officer in the kinds of applications in health care and in manufacturing that motivated this design.

Minsky [182][183] introduces a mechanism for communication in distributed system called "Regulated Interaction (RI)" (previously called "law-governed interaction"). The central concept of this mechanism is a formal and enforced set of rules called a "law.” The approach taken in this work supports heterogeneous systems with disparate coordination policies—much like the approach taken in this thesis—although in a way that is more property-based rather than relationship-based.

5.0 Conclusion

In this thesis, we have shown how the current Internet infrastructure can be augmented by another service layer—one for digital rights management—that allows us to talk about such high-level objects as “contract/agreement,” “obligation,” “right,” and “person”—and provide a platform for structured relationship management that enables us to deal with access to information, privacy and security, etc. as the ancillary of such relationship management.

The Thesis in this Thesis

Our basic hypothesis in the beginning was that with additional software infrastructure, we can unify rights/relationship management from a user-centered perspective, and we can achieve better end-to-end integration than current approaches that do not have access to structured interfaces to relationship information.

This thesis defines a programmable service layer for rights/relationship management and demonstrates a prototype relationship manager application, RManage, which enables users to uniformly manipulate relationships and their properties, and which provides full end-to-end integration for RManage and FIRM-compliant network services to share control and state information. The examples covered by the RManage implementation include various kinds of subscriptions and licenses, as they are widely used in current practice.

The question that we will address in this section is how complete the infrastructure is. This requires a classification of the relationships being supported, so that new kinds of relationships can be accommodated by the FIRM framework. A large part of the answer lies in the programmability of FIRM. FIRM is not a formalism, but a set of object interfaces/APIs with associated definitions about how they can be used. Concrete FIRM-compliant systems will implement some of these interfaces, and use others to interact with existing network objects. The FIRM toolkit that we have implemented facilitates application developers in providing new functionality. For instance, they can subclass implementations of existing objects, such as a site license, and override specific behavior, such as the way in which a payment obligation calculates the amount that is due.

By virtue of being a programmable platform, implementations can add new types of control behaviors and state management functions in a FIRM-compliant way, and go beyond the example cases that we have discussed in the previous chapters. FIRM essentially only guarantees a basic common denominator for the exchange of structured relationship information, but it does not restrict application developers, including forms designers, from providing additional functionality.

Of course, the formats of the attribute models used will have to be defined. This will have to be dealt with much in the same way as done by EDI, which provides an extensive set of data structure definitions for various domains of application. The first-class attribute models used by FIRM provide a systematic mechanism of representing such data structures in a way that can leverage EDI, without being restricted to the assumptions that such systems make about message exchange and communication protocols.

At the next level of detail, FIRM’s specific choice of object interfaces and interactions does not constrain forms designers from being able to implement the kind of control behavior that they want to express. The main effect of FIRM is to allow communication

to shift to a format that uses open interfaces. In principle, one could use a FIRM rights object, for example, in a way that only makes use of the "RequestExercise" method defined by FIRM, with all of the access-control functionality being dealt with as part of this method. Note that this is possible since any method can manage its own state, open its own network connections, etc. We can easily see that such an extreme implementation can realize arbitrary functionality—even though we end up with the old dilemma in this case where much of the state of a relationship is made inaccessible to programmatic access. The second part of the answer is that FIRM only forces application developers to be clear in their conceptualizations about how the various objects that they use relate to concepts that make sense from a contract law perspective. If an application developer cannot answer the question what the obligations of a certain contract are, or what kinds of promissory conditions need to be satisfied to make a right effective, then the potential inability to implement such constructs is justified.

Finally, let us reconsider the issues of security and efficiency. As we have pointed out in the main part of this thesis, FIRM relies on a number of underlying services for security and trust management. The design in this thesis suggests that commpacts be professionally managed on the network by commpact managers. The security considerations for these services are largely the same as for any other network service. Furthermore, commpacts should only be instantiated with commpact managers that are trusted, of course. The necessary trust preferences are part of every e-person's data, as defined by FIRM in the Appendix of this thesis.

The efficiency of the FIRM service layer depends largely on the behavioral complexity that one would like to realize. As we pointed out in the previous chapter, for simple usages that are already possible with current Web servers, for instance, the main difference that FIRM introduces is that of a "published" authorization-request interface, rather than only a server-internal request interface. For more complex usages, the main overhead created by FIRM in the context of integrating with legacy rights-management systems is that of an additional wrapper that maps the legacy protocol into FIRM-compliant structures.

Lessons from the RManage Implementation

We have implemented the FIRM rights management service layer as one of the five service layers of the Stanford Infobus [253] in the context of the Stanford Digital Libraries project. The implementation was based on distributed objects in Java and Python, using the CORBA implementation of one of the our industrial partners (Xerox PARC's ILU system [246]). The sample RManage "relationship manager" application of FIRM that we have prototyped in addition made use of the DLITE user interface toolkit [249]. The testbed of the Stanford Digital Libraries project provided the sample services for which we implemented FIRM-based rights-management solutions.

From the beginning, the purpose of the prototype implementation was primarily to validate the coherence and feasibility of the ideas and concepts suggested in this thesis. While the prototype has been successful in this regard, several issues became evident that suggest that the same implementation design would not generalize well to a full-scale use. We discuss some of these issues below. Note that since the Stanford testbed, and RManage in particular, has not been deployed to a significant number of actual users, we do not have any data on issues related to the usability, maintainability, and robustness of RManage.

One of these issues is that CORBA as a distributed object infrastructure, and in particular the Xerox implementation of it that we used, had shortcomings in terms of robustness as well as scope. In particular, while CORBA defines a whole set of fundamental object services (for object persistence, name resolution, object-level security, etc.), these were not available during the period when we implemented much of the system (beginning in 1995). Partly this certainly reflects the early adopter status that we deliberately stepped into, but partly we believe that this also signifies the tremendous relative complexity of CORBA—compared to, for instance, using protocols that are simply layered on top of HTTP.

Another issue is that the specific implementation design that we chose for the prototype was "too object-oriented" and would be unlikely to scale even if one used more sophisticated implementation environments, such as the CORBA-based Web application server development environments that are now available from such companies as NetDynamics, KivaSoft, and others: Our current implementation uses objects for almost everything, including obligations, rights, and persons. The number of such objects thus easily ends up being quite large in realistic applications, and this quickly becomes infeasible, especially given the resources that each of such an object requires (in terms of memory, for instance). In other words, a somewhat different implementation strategy is needed that is centered around a database to store objects' properties and that uses a transaction monitor for load balancing and for managing the availability of objects whenever they are requested.

Finally, at another level, the services and usages that ended up being available as part of the testbed of the Stanford Digital Libraries project turned out to be largely services that are freely available on the Web, such as the Altavista search service—accessed through proxies with additional processing, though, rather than the Web interface. In other words, the use of rights management and payment was significant only for a few services, such as the Dialog databases. Note though that most Web services are free only to the extent that it is guaranteed that the banner ads that are part of the result pages are visible to viewers. If one uses Altavista, for instance, via a proxy then a license is required in general, and, with FIRM, we were able to accommodate this usage and enable the various broker services of the Stanford Infobus to use such Web services based on site licenses rather than based on an advertising model.

Second-Order Usages

In this thesis, we have primarily examined a range of first-order usages for RManage/FIRM, such as service licensing, various forms of subscriptions, agreeing on the use of personal information, etc. While we have focused on such immediate usages of a computational contracting infrastructure, there is clearly a whole set of second-order usages that we have not begun to explore. These second-order usages make use of the basic contracting infrastructure to implement sophisticated negotiations, and transactions. In particular, consider that e-person implementations can realize complex bidding schemes based on the computational structures that FIRM defines. Since smarter e-person implementations will be highly sought after, we can even imagine a market for e-person "plug-ins" that optimize the negotiation behavior of one's agent viz-a-viz the capabilities of other e-persons.

Outlook

FIRM was originally motivated by a mixed set of issues related to access control and privacy that appeared mostly in the context of people's use of the Web. We identified the absence of a more structured way for sharing control state as the underlying common problem. Effectively, the first wave of Internet technologies had transferred client/server notions into a peer-to-peer environment, but this was not fully adequate.

The design in this thesis provides new infrastructure that addresses these issues by providing a platform for structured relationship management. The application scenarios that motivated us in the beginning can be dealt with by this design. However, we now believe that a FIRM-like infrastructure will not first come to bear on these kinds of usages. It is difficult to assign a clear return-on-investment for consumer-oriented Internet applications of such an improved infrastructure in the near-term future. Consider that while privacy issues are much talked about, it is not clear how much people really value it.

However, another class of application scenarios has recently become evident, and we believe that these will ultimately drive the adoption of a FIRM-like infrastructure. This class of usages takes place in corporate Intranets, which are already typically more tightly integrated and based on a richer platform than the general Internet. In the first wave, enterprises got equipped with Intranets, including ways of automating procurement, enterprise resource planning, etc. In the second wave now, we increasingly run into the question of how external collaborators can be tied into the corporate information infrastructure, that is, how one organization with its Intranet can work together with another organization, such as a partner or a supplier, that also has its own Intranet. But this is exactly the question of how to manage long-standing relationships and facilitate collaboration in a way that takes into account the presence of various boundaries of control.

Two reasons favor FIRM in this environment. One is the significant return-on-investment in the area, that will serve as a driving force. The other is that the base infrastructure in Intranets is already better than on the general Internet. (Distributed objects, for example, have already made inroads in Intranets, while the general Internet is still one or two steps away from this level of complexity.) In other words, we expect that FIRM's concepts will first find their place in helping corporate infrastructure to go beyond Intranets.

Appendix: Specification of the FIRM Rights Management Service Layer

1.0 Overview

The Stanford Framework for Interoperable Rights Management (FIRM) defines a programmable rights management service layer for the Internet that supports a wide variety of applications both in terms of the object representations that it supports and in terms of the transaction protocol.

FIRM is designed to only require participants to agree on a thin, domain-independent interoperability core in order to be able to fully leverage the FIRM infrastructure. By providing a plug-in mechanism, FIRM itself does not need to be extended to be of use in new domains and for new usages; it can be kept simple in its core while allowing application developers to program new kinds of control behaviors for new domains and usages.

This approach allows us to develop a shared ("infra") structure that incorporates protocols and services that are useful for more than one application, that is, we have facilities that can be used to negotiate privacy preferences as well as to negotiate "smart contracts" (e.g. subscriptions and licenses to services), or to manage security, etc. Thus, costs can be amortized over a larger number of usages, and the pay-off is bigger than in developing many disparate special-purpose systems.

The idea for identifying infrastructure components in FIRM is to carefully separate concerns: to limit its specification to a computational reification of only generic principles of rights/relationship management—and to factor out any domain-specific aspects into plug-ins that are kept outside of the type system of the specification. FIRM specifically

identifies contract law as the body of material from which it draws the generic concepts and principles that make up its infrastructure component. Note that contract law can be seen as fundamentally nothing else than a body of concepts and principles that describe the shared structure of rights relationships in a generic way.

FIRM: Two Parts

In other words, rather than having a single-level rights management standard, we have an extensible, two-partite framework:

- a generic (domain-independent) specification that defines a common rights language object model. This model represents the shared structure of different kinds of rights relationships that one might have, and provides the transactional infrastructure for negotiating contracts, exercising contract rights, etc.; and
- a format for declaring domain-specific rights vocabularies that can then be contributed as "plug-ins" to the basic rights management infrastructure. Sample rights languages that can be expressed in this way include the Unix file access rights language, Xerox' DPRL, the EDI standard message types, or specific privacy rights languages.

This appendix describes, in Sections 2 and 3, the corresponding two parts that complement each other to make up FIRM:

- *the FIRM Common Rights Language Object Model:* an interface specification that describes how generic concepts and principles from contract law are reified digitally.
- *the FIRM Object Attribute Models (FOAMs):* a standard format for defining media-specific or domain-specific rights vocabularies, that is, a format for first-class attribute models that define with which attributes FIRM objects represent their state (for use by "agents," etc.).

To understand the separation, consider that FIRM's common object model standardizes on everything that is generic across different domains and cases (and nothing else). For instance, the object model will take up the fact that a contract is between two or more parties and that it is about a set of promises that become effective once the contract has been accepted and all the prerequisite conditions have been fulfilled (contract law). The FOAM attribute models on the other hand will take up domain-specific conceptualization such as the fact that a certain payment obligation is on a per-use basis, that there is a print right that counts the number of copies made (rather than the usage time), etc.

Simplicity, Extensibility, and Distribution

Using a two-level standard that separates specific from generic elements (much like the MIME framework) has many advantages in terms of simplicity, extensibility, and distribution of authority. For instance, by defining wrapper interfaces, FIRM provides a way for legacy implementations (such as payment processing systems) to be tied into the infrastructure. Moreover, as a relationship-based (rather than a property-based) framework, the system is inherently extensible and decentralized: Any party can contribute new rights vocabularies (say, for new domains or new media) by simply "publishing" corresponding rights attribute models. Such extensibility reduces complexity in that components for additional domains/usages only need to be "loaded" on an as-needed basis. In particular, this means that only minimal coordination is required among participants, enabling decentralized administration for everything but the interoperability core. In Section 4, we shall give a number of examples from the RManage prototype implementation of FIRM that demonstrate such plug-ins and wrappers.

2.0 The FIRM Common Rights Language Object Model

2.1 Survey

Before we give a detailed specification, we describe the basic FIRM objects and their properties, as well as the protocol by which clients can directly interact with FIRM objects (the Access-Control User Dialogue Protocol). We also describe which specification language we use.

Basic Objects

In FIRM, contracts, contract parties, rights vocabularies, and even rights and obligations are represented as first-class objects, each with an appropriate set of transactions that can be performed on them. In the following, we give a brief survey of the basic objects of the FIRM specification. The naming convention is that the names of object classes start with a 'C', while the names of simple types start with a 'T'.

The top-level encapsulation of control information is that of a contract object (*CCommpact)*. Commpacts consists primarily of a set of promises (*CPromise*) between two or more parties (*TParty*). Commpacts are effective if they have been mutually agreed upon and if all prerequisite conditions (the 'ConditionsPrecedent' of commpacts and promises) are satisfied.

Each contract party participates in the relationship in a certain role (*TPartyRole*). Such roles are characterized by a constraint on who can fill the role ("all persons whose citizenship is USA"), and possibly a set of the instances of persons (*CEpers*) that are currently filling the role (that is, the finite list of all those who accepted the offer). There can be two or more roles for one commpact, and each role can possibly be filled by more than one party instance (except for the 'offeror' role which is always only one specific party). A commpact has an 'AboutItems' constraint that characterizes which items it is about.

Promises (*CPromise*) can be either rights (*CRight*) or obligations (*CObligation*). Rights are promises that allow promisees to perform certain actions; obligations are promises that require their promisors to perform certain actions. Promises might have a promissory condition ('PromissoryCondition') that specifies which conditions need to be in place before the promise becomes effective. In particular, this can of course include a requirement such as the fact that certain obligations or assertions are satisfied. Promises also have a 'ConditionSubsequent', that is the set of contingency promises that would become effective if the contract is terminated. Among the generic actions that can be performed on promises are methods for them to be waived, transferred, etc. Transferring an obligation is different from just allowing other people to fulfill it (which might be possible); in the former but not in the latter case, it becomes embedded into a new commpact.

Most importantly, of course, rights can be declared to be exercised, and obligations can be declared to be fulfilled. Moreover, there can be additional "trigger" methods that provide further computational integration: For example, fulfilling a payment obligation could trigger an actual payment transfer, upon whose completion the obligation would then be declared to be fulfilled. In other words, declaring fulfillment and initiating fulfillment are two separate actions on an obligation (with the analogous applying for

rights objects). Obligations can register a time with the manager of the commpact of which they are a part. This defines the time at which the objects wants to receive a ‘wakeup’ call. For example, a payment obligation that is due on a certain date could register this time, and then, when woken up, it can decide to fulfill a payment obligation right before it is due.

Also, each object has a method by which one can obtain a description of its most recent state. There is also a method by which one can obtain a generic reference to the object (via a ‘SelfRef’ method that returns types such as *TEpersRef* or *TFormRef*) that contains both a handle to its current address as well as a persistent name to it (a designator that is valid in the long term, but that might have to undergo some name lookup mechanism to resolve it into an address).

Objects and Types Defined by FIRM

In total, FIRM defines the following objects (with their inheritance relation indicated in Figure 33):

- *CFIRMObject:* the most generic rights management object
- *CCommpact:* the contract object
- *CPromise:* the promise object
- *CRight:* a specific type of a promise object
- *CObligation:* a specific type of a promise object
- *CEpers:* the person object
- *CHomeProvider:* the object manager of person objects
- *CCommpactManager:* the object manager of contract objects

Furthermore, the following objects are used in FIRM that are not specific to rights management:

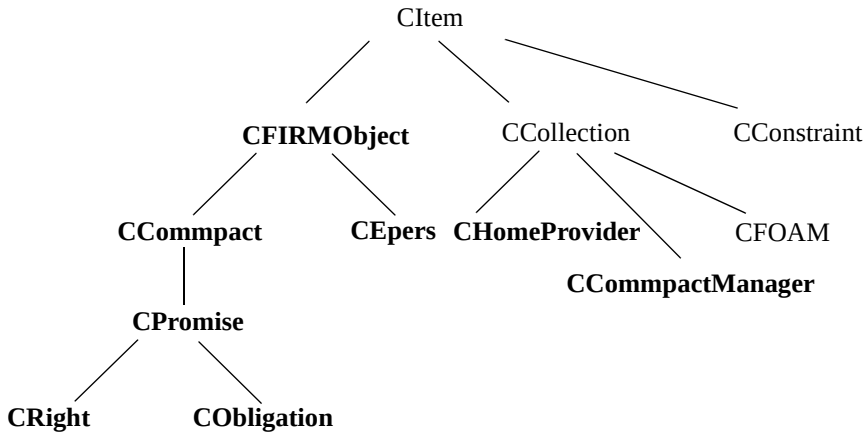
- *CItem:* a generic information item
- *CCollection:* a general collection object that manages items
- *CFOAM:* an attribute model (just a collection of items that define attributes)
- *CConstraint:* a query/constraint object

Finally, the following record types are used in the FIRM specification:

- *TCommpactStatus:* the status of a contract object
- *TPromiseStatus:* the status of a promise object
- *TFormRef:* a way of referencing a contract form
- *TEpersRef:* a way of referencing a person object
- *TParty:* a way of describing the requirements and instances of a certain contract role
- *TPartyRoleName:* the name of a certain contract role (e.g. “Subscriber”)

We also use the type *‘TString’* to specify a general data string (not defined here).

FIGURE 33. FIRM Object Hierarchy: Objects that are not specific to FIRM are not in bold face font.



The Access-Control User Dialogue Protocol (AUPD)

Each FIRM object can be directly manipulated via the Access-Control User Dialogue Protocol (AUPD). This manipulation is interactive; it includes not only interactions such as obtaining a human-readable textual description of a FIRM object, but it also includes interactions such as obtaining a description of a set of options that a FIRM object carries for a certain attribute (e.g. “Choose privacy option: o Your address will be forwarded to interested parties; o No personal information will be used outside of this relationship”), a way of selecting one of the options, and a way of setting it in the FIRM object directly.

A MIME type is used to designate the protocol that is used by a given object for this purpose. In the simplest case, this will be just based on the standard Web conventions for defining “(HTML) forms” and for transferring and parsing the submitted results of such a form (the URL encoding of parameters as used in CGI scripts according to MIME type application/x-url-encoded). However, different user interaction protocols can be defined in principle.

Specification Language

As a specification language, we use CORBA’s “Interface Definition Language” (IDL). IDL defines class interfaces in a cross-platform and language-independent way; it can be viewed as an object system specification language, that is, it specifies the objects including their methods and their instance variables (but it does not define specific implementation issues). In particular, we will be using a variant of IDL here: the ISL (“Interface Specification Language”) of Xerox PARC’s CORBA implementation, ILU [246].[1]

1. ILU is used as a distributed object infrastructure for the testbed of the Stanford Digital Library project as part of which FIRM has been prototyped; the languages used in the testbed to write specific object implementations are Python. C++, and Java.

As with any interface specification, FIRM specifies (in a language- and platform-independent way) the abstract interface to computational (rights management) objects. A specific rights management system will then provide a concrete implementation of this interface, using a specific programming language and a certain environment. For example, the RManage relationship manager application outlined in the main part of this thesis is a prototype implementation of the FIRM interface in Python and Java, providing implementations for contracts such as group licenses, subscription contracts, etc. (cf. also Section 4.0 of this appendix for further detail). Note that it is then the underlying distributed object model that makes various such implementations interoperate seamlessly based on the interface specification and the transactional conventions.

2.2 Specification

```
(*
 * The Stanford FIRM Rights Management Object Model
 *
 * Revision: 3.0
 * Author: "Martin Roscheisen" <rmr@cs.stanford.edu>
 * Date: 1997/04/23 00:59:49
 *)

INTERFACE FIRM
  IMPORTS
    SMA (* the Stanford Metadata Architecture *)
  END;

(*********************************************************)
(* GENERAL                                               *)
(*********************************************************)

(* Items, Collections, Constraints, and Attribute Models *)

  (* For items, collections, constraints, and FOAMs, a simple sam-
     ple interface is given here for purposes of exposition. For
     more versatile interfaces for these objects, see the work on
     the DLIOP and SMA service layers referenced above. For items,
     see also the property service defined by the OMG. *)

  (* - Items: the most generic form of an information object, from
     which representations for other objects (such as documents,
     persons, services, etc.) are derived. Items are essentially
     just a set of properties, that is, a set of attribute-value
     pairs, where the value can be of any type. *)
```

CItem

```
TYPE CItem = OBJECT OPTIONAL
  SUPERCLASS SMA.CItem
  METHODS

    ListPropertyNames(): SEQUENCE OF TString
           RAISES NotAuthorized END,
    GetPropertyValue(name: TString): TAny
           RAISES NoSuchItem, NotAuthorized END,
    SetPropertyValue(name: TString, value: TAny)
           RAISES NotAuthorized END,
```

```
  END;

  (* - Collections: a notion of a grouping of such items. Such
     collections might be constrainable, that is, one can obtain a
     subset of items fulfilling a certain constraint/query. *)
```

CCollection

```
TYPE CCollection = OBJECT OPTIONAL
  SUPERCLASS SMA.CCollection
  METHODS
    ListItemNames(constraint: CConstraint): SEQUENCE OF TString
           RAISES NotAuthorized END,
    GetItem(itemID: TString): CItem
           RAISES NoSuchItem, NotAuthorized END,
    AddItem(itemID: TString, item: CItem)
           RAISES NotAuthorized END,
    RemoveItem(itemID: TString)
           RAISES NoSuchItem, NotAuthorized END,

  END;

  (* - Constraints: the ability to express a constraint on values
     of attributes of a certain attribute model. The exact defini-
     tion of such a language is an orthogonal issue; we can basi-
     cally use any sufficiently rich language here. In particular,
     we can use the same language that is also used for formulat-
     ing search queries (e.g. a simple Boolean query language with
     procedural attachment). *)
```

CConstraint

```
TYPE CConstraint = OBJECT OPTIONAL
  SUPERCLASS SMA.CQuery
  METHODS
    Evaluate(): Boolean
           RAISES NotPossible, NotAuthorized END,
    GetConstraintString(): TString
           RAISES NoSuchItem, NotAuthorized END,
    SetConstraintString(c: TString)
           RAISES NotAuthorized END,

  END;

  (* - FOAMs: Attribute models as first-class objects according to
     the Stanford metadata architecture (SMA). In terms of their
     interface, FOAMs are just collections of attribute definition
     items. *)
```

CFOAM

```
TYPE CFOAM = OBJECT OPTIONAL
  SUPERCLASS CCollection (* of CItems *)
  METHODS
    GetFOAMName(): TString
           RAISES NotAuthorized END,

  END;
```

```
(*************************************************************)
(* FIRM OBJECTS                                              *)
(*************************************************************)

  (* FIRMObjects are the most generic form of an object in FIRM.
     They behave like basic information items, but they also have
     state that can be accessed in a structured way. *)
```

CFIRMObject

```
TYPE CFIRMObject = OBJECT OPTIONAL
  SUPERCLASS CItem
  METHODS

  (* Self-Descriptive Properties *)

  (* A way of referencing the FIRM object persistently *)
     SelfRef(): TFIRMObjectRef
            RAISES NotAuthorized END,

  (* A name of the FIRM object. E.g. "Subscription Contract" *)
     Name(): TString
            RAISES NotAuthorized END,

  (* Different textual descriptions of the FIRM object. There is
     both a short description in plain text that can be inserted
     in the descriptions that are synthesized by other FIRM
     objects (e.g. a commpact can include a list of the descrip-
     tions of its promises). Then there is a full description that
     can include interactive features such as option fields that
     can be selected, etc. The default MIME type is 'text/html'.
     For a commpact, the full description is essentially just like
     the text of conventional legal contracts. *)
     ShortDescription(): TString (* a plain short paragraph *)
            RAISES NotAuthorized END,
     Description(audp: TMime): TString (* a full description *)
            RAISES InvalidInfo, NotAuthorized END,

  (* A callback method for processing the client actions for the
     customization options of a FIRM object. Customization options
     include the ability to fill in specific numbers (e.g. the
     number of times a newsletter appears per week), to select
     options (e.g. which of two delivery modes is preferred,
     etc.). The format for this Access-Control User Dialogue Pro-
     tocol is specified in a MIME type. Typically, this will be
     just the standard HTML forms/cgi result values conventions of
     returning results for user choices. *)
     SetCustomization(audp: TMime, result: TString)
            RAISES InvalidInfo, NotAuthorized END,

  (* Access to (Domain-Specific) State Information *)

  (* FIRM objects such as commpacts and promises are stateful;
     this state usually depends on the specific domain at which
     they are targeted (e.g. "number of searches allowed" in a
     search contract, etc.). *)
```

```
  (* A pointer to the attribute model used (e.g. the "search
     engine rights model" or the "printer model") *)
     GetStateAttributeModel(): CFOAM
            RAISES NotAuthorized END,

  (* An attribute set containing the attributes and their values
     according to the above attribute model *)
     GetState(): CItem
            RAISES NotAuthorized END,

  END; (* of CFIRMObject *)

(*************************************************************)
(* COMMPACTS — Contract Objects Representation               *)
(*************************************************************)

  (* Commpacts ("communication pacts") are the object representa-
     tion of various kinds of agreements including contracts. *)

  (* The Main Contract Object *)
```

CCommpact

```
TYPE CCommpact = OBJECT OPTIONAL
  SUPERCLASS CFIRMObject
  METHODS

  (* Access to Commpact Components *)

  (* A contract is a set of enforceable promises between two
     parties. It specifies about which objects it is, which condi-
     tions are prerequisite for a valid formation, and what the
     general terms and conditions are. The following methods give
     access to these components. *)

  (* The (standard) form on which it was originally based. One
     can always obtain the base form of a commpact via this
     method. *)
     BaseForm(): TFormRef
            RAISES NotAuthorized END,

  (* The current status of the contract (e.g. "offer") as defined
     below *)
     Status(): TCommpactStatus
            RAISES NotAuthorized END,

   (* The contract parties as defined below. The specification of
     a contract party includes a constraint on who can fill a cer-
     tain contract role and the party/parties who are actually
     filling it. *)
     Parties(): SEQUENCE OF TParty
            RAISES NotAuthorized END,

  (* The conditions that have to be satisfied before the contract
     formation is considered to be completed. Note that often
     there is a choice whether to add a condition as a precedent
     condition here or whether to include it as a promissory con-
```

```
   dition as part of a promise-—with somewhat different ramifi-
   cations (e.g. in the former case the contract will not have
   to be cancelled in case of non-fulfillment, but other prom-
   ises will also not be available for use). *)
   ConditionsPrecedent(): CConstraint
          RAISES NotAuthorized END,

(* A set of promises (that is, rights and obligations). *)
   Promises(): CCollection (* of promises (CPromise) *)
          RAISES NotAuthorized END,

(* A constraint that designates the items about which the con-
   tract is. Note that this constraint can easily designate an
   infinite number of items, including non-existing ones (e.g. a
   subscription contract about news issues appearing within the
   next year). *)
   AboutItems(): CConstraint (* what the contract is about *)
          RAISES NotAuthorized END,

(* The general terms and conditions of the contract. For exam-
   ple, these might include the more general kinds of terms that
   would be often printed on the back of legal contract forms.
   The stipulations in this text take up all those policies that
   the attribute model of the contract does not formalize. In
   particular, this includes policies that go beyond what can be
   dealt with by a commpact's code implementation (e.g. conflict
   resolution procedures, etc.). *)
   TnCs(): TString (* textual description *)
          RAISES NotAuthorized END,

(* Transactions *)

(* There are operations to support each of the two phases/modes
   in which a contract lives:
   - "negotiation mode": First, a contract needs to be set up
   and mututal assent about it has to be established.
   - "performance mode": Then, it can be used to give a mandate
   to those actions that fall within the action space that it
   constrains. *)

(* Methods for Contract Creation *)

(* For any commpact, one can obtain the basic template ("form")
   it is based on.Every commpact is derived from one of such
   contract forms.These forms were defined by an appropriate
   "forms designer" authority. One can take and instantiate such
   a form into a "draft" commpact, which can then be customized
   appropriately.[If there is a newer version for this template
   than the one that was used for the given commpact instance,
   then the newer version will be returned here (but the form
   obtained via the BaseForm method will still refer to the old
   version).] *)
   GetForm(newOfferor: TEpersRef): CCommpact
          RAISES NotAuthorized END,
```

```
(* A method for adding promises to a commpact instance. In par-
   ticular, certain commpact actions might create new promises
   that then need to be added to its set of promises (e.g. prom-
   ises from the condition subsequent). In particular, if the
   added promise is an obligation, then this method will immedi-
   ately activate it ("wakeup") to allow it to register a future
   activation time or to indicate that it needs to be fulfilled
   immediately (such as when exercising a certain search right
   creates an obligation to filter or watermark its results
   before they are returned). *)
  NewPromise(name: TString): CPromise
         RAISES NotAuthorized END,
  ASYNCHRONOUS RequestAddPromise(promise: CPromise),
  DoneAddPromise() (* callback *)
         RAISES NotAuthorized END,

(* Methods for Negotiation Mode *)

(* Once drafts are customized, they can be declared to be an
   offer. Such offers can then be accepted or rejected (by who-
   ever qualifies as an offeree), or be revoked (by the off-
   eror). A validly formed commpact ("effective") can then also
   be terminated; such termination might in turn require other
   "subsequent conditions" to be performed (which is why there
   is technically an asynchronous variant for this method). *)
  ASYNCHRONOUS RequestDeclareDraftAnOffer(actor: TEpersRef),
  DeclareDraftAnOffer(actor: TEpersRef)
         RAISES InvalidStatus, NotAuthorized END,
  ASYNCHRONOUS RequestAcceptOffer(actor: TEpersRef),
  DeclareOfferAccepted(actor: TEpersRef): CCommpact
         RAISES InvalidStatus, NotAuthorized END,
  ASYNCHRONOUS RequestRevokeOffer(actor: TEpersRef),
  DeclareOfferRevoked(actor: TEpersRef)
         RAISES InvalidStatus, NotAuthorized END,
  ASYNCHRONOUSRequestRejectOffer(actor: TEpersRef),
  DeclareOfferRejected(actor: TEpersRef)
         RAISES InvalidStatus, NotAuthorized END,
  ASYNCHRONOUS RequestModifyOffer(actor: TEpersRef, extent:
         CItem),
  DeclareOfferModified(actor: TEpersRef, extent: CItem): CCom-
         mpact
         RAISES InvalidStatus, NotAuthorized END,
  ASYNCHRONOUSRequestRenegotiate(actor: TEpersRef),
  DeclareAcceptRenegotiate(actor: TEpersRef)
         RAISES InvalidStatus, NotAuthorized END,
  ASYNCHRONOUS RequestTerminateCommpact(actor: TEpersRef),
  DeclareCommpactTerminated(actor: TEpersRef)
         RAISES InvalidStatus, NotAuthorized END,
  ASYNCHRONOUS RequestCompleteCommpact(actor: TEpersRef),
  DeclareCommpactCompleted(actor: TEpersRef)
         RAISES InvalidStatus, NotAuthorized END,

(* Methods for Performance Mode *)
```

```
    (* A simple method for finding a promise. Of course, clients can
       always also directly interface with the whole collection of
       promises themselves; the following method is intended to make
       the frequent case simple and allow for server-side optimiza-
       tions. Note that there is no separate 'authorize' method—
       rights are just exercised directly. *)
       FindPromise(holder: TEpersRef, name: TString): CPromise
              RAISES NoSuchItem, MultipleItems, NotAuthorized END,

  END; (* of CCommpact *)

  (* Auxiliary Types for Commpacts *)

  (* States in which a commpact can be. The transitions among
     these states are defined by the corresponding transactions
     (e.g. declaring it an offer turns a draft into an offer). *)
```

TCommpactStatus

```
TYPE TCommpactStatus = ENUMERATION
  (* The inital commpact template from a forms provider *)
     Form,
  (* A status to designate a variety of intermediate situations—-
     such as when we have a revoked offer or when someone is work-
     ing on the customization of a form (in which case we have
     more than a form but not yet an offer). *)
     Draft,
  (* A valid offer that is ready to be accepted *)
     Offer,
  (* An effective agreement (i.e., an accepted offer with forma-
     tion complete *)
     Effective,
  (* A terminated relationship *)
     Terminated
  END; (* of TCommpactStatus *)

  (* A way of referencing a (standard) commpact form, that is, the
     template on which commpacts are based: by naming its identi-
     fier (e.g. "Standard License for Web Search Engines"), the
     version number, and the authority who originally provided it.
     Optionally again, a current (direct) address of an object
     instance of it. *)
```

TFormRef

```
TYPE TFormRef = RECORD
     formName: TString,
     version: TString,
     formProvider: TEpersRef,
     form: CCommpact (* optional *)
  END; (* of TFormRef *)

  (* A way of characterizing a contract party: by designating the
     role that this party plays as part of the contract (e.g.
     "subscriber"), any constraint that applies to filling this
     role (e.g. "student status required"), and the set of e-per-
```

```
    sons that are actually currently filling this role. In the
    most typical cases, a contract will just have two party
    roles, with each one e-person filling one of the roles (off-
    eror and offeree).  However, in some cases we might have some
    third-party role in addition, and in other cases, we can
    allow multiple e-persons to fill one and the same role if the
    contract's state information does not depend on the identity
    of the filler. *)
```

TParty

```
TYPE TParty = RECORD
  (* A "local name" for the contract party *)
     roleName: TPartyRoleName,
  (* A constraint on who can fill this role *)
     constraint: CConstraint,
  (* The e-persons that actually accepted to fill this role *)
     instances: CCollection (* of e-persons (CEpers) *)
  END; (* of TParty *)

  (* A role name gives a way of specifying contract templates
     without having to name/know the specific party at specifica-
     tion time; they can then be used to specify which party has
     holds which promises, etc.  In other words, these names func-
     tion much like local variable declarations. *)
```

TPartyRoleName

```
TYPE TPartyRoleName = TString;

(*****************************************************************)
(* PROMISES                                                      *)
(*****************************************************************)

  (* Promises are the core component of a contract as a set of
     enforceable promises.  The two main subtypes are rights and
     obligations.  Promises are represented themselves as first-
     class objects within the context of a commpact. *)

  (* The Main Promise Object Representation *)
```

CPromise

```
TYPE CPromise = OBJECT OPTIONAL
  SUPERCLASS CFIRMObject
  METHODS

  (* Access to Promise Components *)

  (* A promise is a right or an obligation held by one of the con-
     tract parties. It can be only about a subset of the objects
     that the contract talks about. *)

  (* A local reference to the party who promises it *)
     Promisor(): TPartyRoleName
            RAISES NotAuthorized END,

  (* In case of multi-party contracts with more than two parties,
     a local reference to the party who is the promisee *)
```

```
   Promisee(): TPartyRoleName
          RAISES NotAuthorized END,

  (* The current generic status of the promise (see below) *)
   Status(): TPromiseStatus
          RAISES NotAuthorized END,

  (* Possibly any conditions that need to be in place before the
     promise becomes effective (e.g. a delivery promise under the
     condition that a payment obligation was fulfilled). *)
   PromissoryCondition(): CConstraint
                          (* on commpact attributes *)
          RAISES NotAuthorized END,

  (* Possibly any kinds of promises that come into place upon ter-
     mination of the contract relationship *)
   ConditionsSubsequent(): CCollection
                           (* of promises (CPromise) *)
          RAISES NotAuthorized END,

  (* Optionally an additional constraint that specifies the subset
     of items which the promise is about—in case it is not about
     all those that are subject of the commpact itself (default
     case). *)
   AboutItems(): CConstraint (* on which items it is about *)
          RAISES NotAuthorized END,

  (* Transactions *)

  (* A method for exercising a right or fulfilling an obligation
     (a generic redirect to the 'exercise' and 'fulfill' methods
     of the CRight and CObligation subtypes, respectively). *)
   ASYNCHRONOUS RequestDeclareExercised(actor: TEpersRef,
          extent: CItem),
   DeclareExercised(actor: TEpersRef, extent: CItem)
          RAISES InvalidInfo NotAuthorized END,

  (* Whether or not the promise is currently effective *)
   CheckValidity(): Boolean
          RAISES NotAuthorized END,

  (* Methods for transferring a promise to another party *)
   ASYNCHRONOUS RequestDeclareTransferred(newHolder: TEpersRef,
          newCpct: CCommpact),
   DeclareTransferred(newHolder: TEpersRef, newCpct: CCommpact)
          RAISES NotAuthorized END,

  (* Methods for waiving a promise (in the negotiation phase) *)
   ASYNCHRONOUS RequestDeclareWaived(),
   DeclareWaived()
          RAISES NotAuthorized END,

  END; (* of CPromise *)

(* Auxiliary Types for Promises *)
```

```
  (* Generic state information for every promise includes whether
     it is exclusive, waivable (during negotiation), transferrable
     (to other parties), whether it is (unconditionally) effec-
     tive, and whether it has already been fulfilled. *)
```

TPromiseStatus

```
TYPE TPromiseStatus = RECORD
    exclusive: Boolean, (* e.g. an exclusive right *)
    waivable: Boolean, (* whether the promise is waivable *)
    transferrable: Boolean, (* whether it is transferrable *)
    waived: Boolean, (* whether it was indeed waived *)
    transferred: Boolean, (* whether it was transferred *)
    effective: Boolean, (* whether it is currently effective *)
    fulfilled: Boolean (* whether it was completed *)
  END; (* of TPromiseStatus *)

  (* Note on case-specific parameterization: Transactions on prom-
     ises are often qualified by domain-specific action parame-
     ters. For example, when fulfilling a payment obligation,
     someone might opt to do this in part only, or when exercising
     a fancy print right, one might want to indicate with which
     resolution to print a document. Whenever applicable, FIRM
     allows these parameters to be passed along to methods as part
     of an argument of type 'CItem'. This type is just an
     attribute set, and the assumption is that it is structured
     according to the attribute model of the promise. In other
     words, when fulfilling a payment obligation that has an
     attribute model containing a 'sumToPay' attribute (with a
     correspondingly defined meaning), we can pass along a value
     for this attribute as part of the 'extent' argument of type
     CItem. *)

(*****************************************************************)
(* RIGHTS and OBLIGATIONS                                        *)
(*****************************************************************)

  (* Rights and obligations are the immediate subtypes of prom-
     ises. [Other promises such as guarantees and warranties are
     considered to be just obligations with corresponding promis-
     sory conditions.] *)

  (* Rights *)
```

CRight

```
TYPE CRight = OBJECT OPTIONAL
  SUPERCLASS CPromise
  METHODS

  (* A method for declaring that a right has been exercised (fully
     or partially). The extent to which it is "exhausted" is inde-
     cated (e.g. "fully" or a certain fraction such as 31 of 1000
     allowed searches).  Promises keep track of these state
     changes in a promise-specific way. *)
    DeclareExercised(actor: TEpersRef, extent: CItem)
           RAISES NotAuthorized END,
```

```
    (* A method for exercising a right (fully or partially). This
       corresponds to an authorization call; rights-specific param-
       eters can be provided as part of the 'extent' argument (e.g.
       the number of searches to allow as part of a request to
       authorize searches). *)
      ASYNCHRONOUS Exercise(actor: TEpersRef, extent: CItem)

  END; (* of CRight *)

  (* Obligations *)
```

CObligation

```
  TYPE CObligation = OBJECT OPTIONAL
    SUPERCLASS CPromise
    METHODS

    (* A method for declaring an obligation fulfilled. Contract par-
       ties or appropriate third parties can declare an obligation
       to be (partially or completely) fulfilled. The promise keeps
       track of who has made which declarations such that inconsis-
       tencies can be detected.  The extent to which an obligation
       is fulfilled uses obligation-specific attributes; this is
       indicated with a corresponding 'extent'-cookie that can have
       any value (following the attribute model of the promise). A
       null value is taken to be "completely fulfilled". *)
      ASYNCHRONOUS RequestDeclareFulfilled(actor: TEpersRef,
            extent: CItem),
      DeclareFulfilled(actor: TEpersRef, extent: CItem)
            RAISES NotAuthorized END,

    (* A method for triggering the actual process of fulfilling an
       obligations (if available): The obligation holder can ini-
       tiate a (complete or partial) fulfillment of the obligation
       (e.g. initiate electronic payment). *)
      ASYNCHRONOUS Fulfill(actor: TEpersRef, extent: CItem),

    (* A way for the promisee or any observing third party to
       request the (complete or partial) fulfillment of the obga-
       tion. *)
      RequestFulfill(actor: TEpersRef, extent: CItem)
            RAISES NotAuthorized, NotAnEffectivePromise END,

    (* An obligation can register a wake-up time with the manager of
       the commpact that it is part of (see the commpact manager
       interface). The following method is then called back. *)
      Wakeup(cookie: TString)
            RAISES NotAuthorized END,

    (* A method for inspecting those other rights or obligations
       that are created by not living up to an obligation (e.g. a
       penalty payment obligation through late payment; a termina-
       tion right through non-delivery, etc.) *)
      GetConsequences(): CCollection (* of CPromise *)
            RAISES NotAuthorized END

  END; (* of CObligation *)
```

```
(*************************************************************)
(* EPERS — Person Object Representations                     *)
(*************************************************************)

  (* An epers/e-person is the object representation of (a role of)
     a person (natural, corporate, etc.).  It is essentially a
     generalization of the notion of a (Unix, AOL,...) "account"
     in that it provides the online represenation of a person with
     a request interface (which is more structured here than with
     the forms of accounts that we currently know). One and the
     same human can have multiple e-persons (e.g. one for private
     and one for a certain organizational function). *)
```

CEpers

```
TYPE CEpers = OBJECT OPTIONAL
  SUPERCLASS CFIRMObject
  METHODS

  (* Standard collections for every e-person. Every e-person has
     at least four default collections: *)

  (* - the commpacts currently held by this e-person, *)
     Commpacts(): CCollection (* of commpacts *)
            RAISES NotAuthorized END,

  (* - the set of public offers from this e-person, and *)
     Offers(): CCollection (* of commpact offers *)
            RAISES NotAuthorized END,

  (* - a notifier as a unified "attention structure" for the e-
     person   (the e-person's "inbox"). *)
     Notifier(): CCollection (* of CItems *)
            RAISES NotAuthorized END,

  (* Methods for telling the e-person whether its user (the real
     person) is online and, if so, where to reach it there: by
     telling it about the current "local resource manager" (LRM),
     that is, an object that knows about how to get to the user's
     screen, etc. *)
     RegisterLRM(lrmName: TString)
            RAISES NotAuthorized END,
     UnregisterLRM()
            RAISES NotAuthorized END,

  (* A way of notifying an e-person of something. For example,
     some party might request an e-person to accept a certain
     commpact offer. For certain types of offers, the e-person's
     default preferences will lead it to accept this offer auto-
     matically (in which case it might not even show up on the
     notifier); in other cases, it might involve more direct user
     feedback. *)
     ASYNCHRONOUS RequestAcceptOffer(offer: CCommpact),
     ASYNCHRONOUS RequestRejectOffer(offer: CCommpact),
     ASYNCHRONOUS RequestRevokeOffer(offer: CCommpact),
     ASYNCHRONOUS RequestModifyOffer(offer: CCommpact),
```

```
    ASYNCHRONOUS RequestTerminateCommpact(cpct: CCommpact),
    ASYNCHRONOUS RequestRenegotiateCommpact(cpct: CCommpact),

  (* Optional callbacks for a number of negotiation transactions
     that an e-person might want to be notified about. *)
    ASYNCHRONOUS OfferDeclaredRejected(cpct: CCommpact),
    ASYNCHRONOUS OfferDeclaredRevoked(cpct: CCommpact),
    ASYNCHRONOUS OfferDeclaredAccepted(cpct: CCommpact),
    ASYNCHRONOUS OfferDeclaredModified(cpct: CCommpact),
    ASYNCHRONOUS CommpactDeclaredTerminated(ccpt: CCommpact),
    ASYNCHRONOUS CommpactDeclaredRenegotiate(ccpt: CCommpact),

  (* A set of preferences that determine some of the automatic
     default behavior of an e-person.  Note that these preferences
     are just obligations held by the e-person. For example, there
     might be obligations that determine which types of commpact
     offers the e-person is to accept automatically, which ones to
     pick to authorize performing a certain action (automatically
     or not), etc. Since these obligations held by the e-person to
     their "outside owner" (the real person), these obligations
     are special in that they are not part of any commpact. *)
    DefaultEpersObligations(): CCollection (* of CObligation *)
          RAISES NotAuthorized END,

  (* A way of automating the choice of which commpact to use by
     default for a certain action (such as, for example, whether
     to use a subscription or a pay-per-search contract for a
     given search engine—assuming that the e-person has both).
     Based on its preference rules (obligations), an e-person
     determines the default commpact that is to be used to autho-
     rize an action. In the simplest case, this will be just based
     on the item considered (e.g. the search engine activated).
     In more complex cases, this might lead to more complicated
     negotiation. *)
    GetDefaultCommpact(action: TString,
          item: TString, ownedBy: CEpers): CCommpact
          RAISES NotAuthorized END,

  (* Authentication. A network login is done by authenticating
     oneself (one's browser's network address, etc.) with respect
     to one's e-person.  A credential signed with the public key
     of the home provider is returned. *)
    Authenticate(password: TString, selfInfo: TEpersRef): TString
          RAISES InvalidAuthentication, NotAuthorized END,

  (* An e-person can refer to another e-person and redirect cer-
     tain requests to it.  For example, instead of replicating
     certain personal information that is constant over various e-
     persons ("roles") that a person might have (e.g. the person's
     height), this information can be kept at one e-person only
     and the others cause exceptions when such attributes are
     requested and redirect to a designated other e-person.  The
     object model is one of "classless inheritance" here. *)
```

```
    GetSuperEpers(): CEpers
           RAISES NotAuthorized END,

  (* A default "personal relationship manager." *)
    GetCommpactManager(): CCommpactManager
           RAISES NotAuthorized END

  END; (* of CEpers *)

  (* A persistent way of referencing and identifying an e-person
     *)
```

TEpersRef

```
TYPE TEpersRef = RECORD
   (* The pseudonym of the e-person *)
    epersID: TString,

  (* The name of the home provider, that is, the manager of the
     person object *)
     homeProviderID: TString; (* object manager name *)

  (* Optionally, a cryptographic guarantee that there is indeed
     the e-person with the above attributes behind the requesting
     network address (e.g. using a PK-signed hash of these three
     information pieces). Such a warrant can be obtained by suc-
     cessfully performing a network login, that is, by providing
     valid authentication information to the home provider and in
     return obtaining a corresponding warrant that is signed by
     the home provider's private key). *)
    warrant: TString,

  (* Optionally, the object's current network address (string
     binding handle) *)
     epersHandle: TString

END; (* of TEpersRef *)

(*****************************************************************)
(* HOME PROVIDER                                                 *)
(*****************************************************************)

  (* E-person objects are managed by an object manager called
     "home provider". Here, a home provider is represented as a
     searchable collection of e-persons.  Often a home provider
     will integrate the following two services (although this is
     not strictly necessary): a default collection manager that is
     able to persistently manage the member e-persons' items and
     collections (as their personal "file space"), and a default
     commpact manager that manages the relationship objects (com-
     mpacts) of an e-person. *)
```

CHomeProvider

```
TYPE CHomeProvider = OBJECT OPTIONAL
  SUPERCLASS CCollection (* of CEpers  *)
  METHODS

  (* Request new e-person to be created with a certain name  *)
```

```
    NewEpers(epersID: TString): CEpers
           RAISES InvalidInfo, NotAuthorized END,

  (* Obtain the home providers public key. *)
    GetPublicKey(): TString
           RAISES NotAuthorized END,

  (* Optionally, a default commpact manager *)
    GetDefaultCommpactManager(): CCommpactManager
           RAISES NotAuthorized END,

  (* Auxiliary methods for finding and removing e-persons  *)
    FindEpers(epersID: TString): CEpers
           RAISES NoSuchItem, NotAuthorized END,
    RemoveEpers(epersID: TString)
           RAISES NoSuchItem, NotAuthorized END,

  END; (* of CHomeProvider *)

(*************************************************************)
(* COMMPACT MANAGER                                          *)
(*************************************************************)

  (* A commpact manager manages/keeps e-persons' commpacts. Spe-
     cial cases of commpact managers are forms providers (who make
     available standard digital contract forms) and personal
     commpact/relationship managers. Commpact Managers can reside
     with a server (conventional access control), but also with a
     client or with a trusted third party (e.g. a rights clearing-
     house). *)
```

CCommpactManager

```
TYPE CCommpactManager = OBJECT OPTIONAL
  SUPERCLASS CCollection (* of commpacts *)
  METHODS

  (* A way of asking the commpact manager to instantiate a new
     commpact based on a certain template  *)
    NewCommpact(actor: TEpersRef, form: TFormRef): CCommpact
           RAISES InvalidInfo, NotAuthorized END,

  (* A way of getting hold of the commpact manager's public key*)
    GetPublicKey(): TString
           RAISES NotAuthorized END,

  (* A way of registering a wake-up call. For example, obligations
     can register such a time with the manager of the commpact
     that they are part of; thus, they can be sure to be activated
     at any required deadline. A cookie can be added to provide
     information on the wakeup context. *)
    RegisterWakeupTime(time: TString, cookie: TString)
           RAISES NotAuthorized END,

  (* Auxiliary methods for finding commpacts and commpact forms.*)
    FindForm(actor: TEpersRef, name: TFormRef): CCommpact
           RAISES NoSuchItem END,
```

```
    FindCommpact(actor: TEpersRef, name: TString): CCommpact
           RAISES NoSuchItem END,

  END; (* of CCommpactManager *)

(*************************************************************)
(* EXCEPTIONS                                                *)
(*************************************************************)

  (* Note: In this specification, only high-level exceptions have
     been explicitly indicated. It is assumed that every method
     can also throw a set of base-level (CORBA) object system
     exceptions such as 'CommunicationFailure', 'InterfaceVer-
     sionMismatch', etc. *)

EXCEPTION InvalidAuthentication;
EXCEPTION NotAuthorized;
EXCEPTION NoSuchItem;
EXCEPTION MultipleItems;
EXCEPTION NotPossible;
EXCEPTION InvalidInfo;
EXCEPTION InvalidStatus;
```

3.0 FIRM's Object Attribute Models

Complementing the common rights object model outlined in the previous section, we have domain-specific rights conceptualizations. These are not taken into the type system of the object model, but they are specified independently. FIRM uses attribute models (FOAMs) to "publish" the attributes with which the corresponding objects represent their state—in a way that is useful for other programs (such as "agents"). Note that FIRM only specifies a format for the attribute models; it does not itself specify any concrete attribute models.

FOAMs give the flexibility of adding rights vocabularies for specific domains on top of an already established basic FIRM infrastructure. In particular, this flexibility is achieved in a way that does not assume any centralized infrastructure (standards institution), and that is extensible over time without the need to modify the basic FIRM specification.

To understand FOAMs, consider the Unix file access rights as a simple example of a rights language. The Unix file access rights define a rights relationship between the owner of a file and other interested parties. Specifically, this relationship is conceptualized by three promises, all rights, namely the well-known rights to read, write, and execute. A FIRM wrapper/proxy that makes these rights available to other FIRM services would take up this fact in a commpact. Then, for each of the rights, we then have a promise attribute model (FOAM) that describes which attributes characterize the state of the promise. In the case of the write right in Unix, this is for instance the last-accessed time. This time will be one attribute of the write-right FOAM. Of course, another attribute is the identity of the holder of the right. But this does not need to be part of the FOAM—since it is already part of the generic structure of a promise.

The meta-data architecture of the Stanford Digital Libraries Project defines a format for first-class attribute models that is suitable for FIRM's attribute models.

3.1 Attribute Models in the Stanford Metadata Architecture

We are assuming in FIRM the availability of the kind of attribute services that the Stanford meta-data architecture provides.

The Stanford metadata architecture is designed to provide an infrastructure that affords interoperability among heterogeneous, autonomous digital library services. It includes attribute model proxies, attribute model translation services, metadata information facilities, and local metadata repositories.

For our purposes here, we are mainly interested in the following characteristics:

- Attribute models are first-class objects that can be referred to and that can be searched for specific attribute definitions.
- Attribute models are thus fundamentally a collection of items, where each item is an attribute definition containing the attribute name, an attribute documentation (a textual description of its meaning), and a definition of the attribute value (using IDL syntax).
- New attribute models can be defined by anyone by providing a collection of new attribute definitions and by publishing it.

3.2 Attribute Models as Domain Plug-Ins for FIRM: FOAMs

As pointed out above, the FIRM common object model specifies only an abstract interface to rights management objects. Any concrete rights management system will provide a specific implementation of this interface that realizes the (domain-specific) behavior in that one is interested in a specific case. For example, someone might provide a Java implementation for a FIRM obligation object that represents a payment obligation, such that triggering the 'fulfill' method on this object would actually lead to a real payment transaction from one bank account to another.

In this way, we are flexible with regard to which domain-specific conceptualization to choose. For example, a print right in one rights language might provide for a counter of the number of times that a document can be printed, the maximum resolution that this right allows a printer to have, etc. If someone considers this insufficient, then this can be extended to include further attributes (possibly inheriting from the previous).

Consider a payment obligation that is characterized by two attributes, namely the amount that was due and the amount that was already paid. To make this explicit, a reference to an attribute model that defines these two attributes will be included in the payment promise object. Thus, anyone can ask the promise object for the attribute model it uses for its state, and then ask the object for specific values of this state (if authorized). Moreover, implementations of FIRM objects will use these attribute models themselves for various purposes. For example, a payment obligation that is based on pay-per-use will have an attribute that describes the price per use, for instance. But to come up with an amount to be paid, it will need to know about the number of uses that are applicable. If embedded in a search contract that contains a search right next to the payment obliga-

tion, it can then obtain this number from the search right (that will count the number of searches for every time it is exercised to perform a search).

All FIRM objects declare which attribute model they use to represent their state. In principle, new attribute models can be defined by anyone, although it is of course to be expected that there will be a certain set of "standard" models for generic cases. Such models would be efficient in terms of reuse and in terms of making available a language by which one can talk about domain-specific issues across different implementations and platforms.

3.3 Sample Attribute Models for FIRM Objects

In this subsection, we examine a few concrete attribute models for sample FIRM object implementations. Note how these FOAMs effectively make explicit the conceptualization that a designer chose for a certain action, promise, or relationship.

For example, it is as part of the attribute model that the designer declares whether a search right is characterized only by a number of searches completed relative to those allowed, or whether it includes a counter of the total number of items returned for all searches, etc.

```
SimpleSearchRightModel::RightsAttributeModel = {
   attr1 = {
       attrName: 'searchBudget'
       attrValueType: 'CARDINAL'
       attrDocumentation: 'Total number of searches allowed'
   }
   attr2 = {
       attrName: 'searchCount'
       attrValueType: 'CARDINAL'
       attrDocumentation: 'Total number of searches done so far'
   }
}
```

The above defines a new attribute model that inherits from 'RightsAttributeModel' (defined elsewhere) and adds two attribute items, each with their attribute name (a string), their value (an ISL specification as a string), and the intended meaning described by a textual description.

Alternatively, someone might define a more complex search right model, which also represents the number of items returned for each search:

```
SimpleItemizedSearchRightModel::SearchRightModel = {
   attr1 = {
       attrName: 'searchHistory'
       attrValueType: 'SEQUENCE OF RECORD
            time: TTime,
            resultSetSize: CARDINAL
            END'
       attrDocumentation: 'History of searches that have been
            successfully completed so far—as a list of structures,
            one for each search, where each structure describes
            the time of the search and the number of items
            returned for this search'
```

```
    }
}
```

As another example, consider the following attribute model that defines case-specific state information for a payment obligation that operates on a per-use (e.g. per search) basis. Note that the holder and the beneficiary of the payment obligation (the e-person that has to pay and the receiver) are already defined as part of the generic structure of a promise.

```
SimplePayPerUsePaymentObligationModel::ObligationModel = {
    attr1 ={
        attrName: 'pricePerUse'
        attrValueType: 'CARDINAL'
        attrDocumentation: 'Price per use in US Dollars'
    }
    attr2 = {
        attrName: 'minAmount'
        attrValueType: 'CARDINAL'
        attrDocumentation: 'Minimum charge to be applied (in USD)'
    }
}
```

Since promises know about the commpact in which they are embedded, we could now have a search license (commpact) that allows people to use a search service, with an obligation to pay on a per-search basis.

In other words, attribute models are used here to complement FIRM in that they provide all the domain-specific state structure that FIRM intentionally abstains from defining; they can be contributed by anyone, and they are managed in a distributed way (as a set of attribute model servers).

FIRM's attribute models provide a format by which other rights languages can be tied into the framework. This includes specific mechanisms such as the Unix file access rights language, but also more extensive standards such as the domain-specific record formats of EDI's standard message types.

3.4 Attribute Models and Interoperability of Heterogeneous Rights Languages

Recall that it is our assumption that there will continue to be a heterogeneous set of rights languages, each generally targeted at a specific design trade-off space. FIRM can be used to address the issue of interoperability across different rights languages. FIRM's interfaces would serve as a common dominator for interoperation, and its FOAM attribute models could be used to "export" the defining properties from specific rights vocabularies and make them available across different implementation languages and platforms via the FIRM interface.

In particular, once there is a basic mechanical interoperability attained in this way, we can achieve a more semantic level of interoperability by additionally providing implementations for services such as the attribute translators that the Stanford metadata architecture provides for.

In Section 4.6, we will give an example of how such interoperability can work for the simple case of the Unix file access rights language.

4.0 Examples of Interface Implementations

The FIRM specification in Section 2.0 is an interface specification that defines the abstract request interface by which objects on possibly different platforms and possibly implemented in different languages can interact. The following demonstrates how a competent "forms provider" can implement and make available new types of contracts in a relatively easy way by leveraging a class hierarchy. These examples of specific implementations are given in Python; they are drawn from the RManage prototype implementation.

4.1 Example Commpact: A Site Licensing Contract

The most basic form of a sample subscription commpact is very simple if we use the RManage toolkit to provide an implementation 'CCommpact' that takes care of all of the generic transactional behavior. Essentially, we only have to provide then some documentation information and create the initial promises (which deal with most of the contract-specific behavior).

```
class SubscriptionCommpact(CommpactImpl.CCommpact):
  def __init__(self):
    self.roleNames = ["Publisher", "Subscriber"]
    CommpactImpl.CCommpact.__init__(self)

    # Add Promises
    prom = self.NewPromise("SearchRight")
    self.AddPromise(prom)
    prom = self.NewPromise("FlatFeePaymentObligation")
    self.AddPromise(prom)
  def Name(self):
    return "Subscription Contract"
  def TnCs(self):
    return "The standard terms and conditions of the retail
       industry are included by reference regarding the return of
        goods and warranties."
  def ShortDescription(self):
    return "A subscription agreement in which publisher agrees to
       deliver the indicated number of issues to subscriber who in
        turn agrees to pay at least quarterly."
```

4.2 Example Customization: Adding a Privacy Choice

Let us now consider that we want to add the choice of a privacy option to the contract, using an HTML-based description. This can be done quite easily by just implementing the following method of the RManage commpact toolkit to the above implementation:

```
  def GetCustomizationOptions(self):
    # Get generic options, if any
    page = CommpactImpl.CCommpact.GetCustomizationOptions(self)
    # Add our specific options to offers
    if self.Status()=="Offer":
       page = page + "<P>Restrict use of address information ?
                 <INPUT TYPE=checkbox NAME=addressPriv VALUE=Y "
              + self["addressPriv"] + "> Y/N"
    else:
```

```
    # If not an offer, just include choice in description
     page = page + "<P>Restrict use of address information? "
            + self["addressPriv"]
  return page
```

This is all that is needed to introduce another state variable (`addressPriv`) to the commpact and provide all the interactive behavior to allow people (or their e-persons) to pick a privacy option when accepting an offer.

4.3 Example Promise: A Payment Obligation

The following Python code sketches the implementation of a specific subtype of a payment obligation based on a flat-fee pricing model. The class from which it is derived itself would be a more generic implementation of a payment obligation, realizing a basic FIRM obligation object. In particular, note how descriptions are generated from state information.

```
class FlatFeePaymentObligation(PaymentObligation):
  def __init__(self):
    PaymentObligation.__init__(self)
    self["flatFee"] = 300
    self["amountPaid"] = 0

  def ShortDescription(self):
    # Generate descriptive text
    page = PaymentObligation.ShortDescription(self)
    page = page + "The following flat fee needs to be paid: "
           + self["flatFee"] + "USD in total.<P> Of this, "
           + (self["flatFee"]-self["amountPaid"])
           + "USD are still due."
    return page
```

This class does not need to provide a 'fulfill' method since we can use by default the one from its parent class (the payment obligation). The fulfillment method there would be implemented to actually initiate a payment transfer, for example. In RManage, this would be done quite easily using the UPAI application program interface. Note, however, that it is in principle not strictly required to also provide the link to actual automatic payment as a fulfillment action; we might as well just assume that payment is done off-line, and only use the DeclareFulfilled method to register new state.

The following additional method would make it possible for offerors to customize the initial flat fee (while drafting the contract offer).

```
  def GetCustomizationOptions(self):
    if self.cpctContext.Status()=="Draft":
      return "<P>Flat fee: <INPUT SIZE=5 NAME=flatFee
             VALUE="+self["flatFee"]+">"
```

4.4 Example Authorization: Allowing Searching with a Search License

Let us consider the following case: We have a Web search engine that can be used by anyone with a license. A client already has one such license (otherwise, it would have to be negotiated first), and wants to conduct a search. The following actions will now result: First, the client sends the search request to the search engine. This request

includes the query and a reference to the contract that the searcher wants to use (the client's e-person might automatically pick one based on the user's designated preferences). Then, the search engine receives this information, and does an authorization along the lines of the following code fragment (at least initially):

```
# Requester: e-person of the searching user
# Params: what kind of search to conduct; e.g. could
#     be the following default parameters
params = {"maxResultSetSize": 100}
# licenseCpct: the commpact that the user wants to use

searchRight = licenseCpct.FindPromise(requester,"Search")
try:
  searchRight.Exercise(requester, params)
except FIRM.NotAuthorized:
  raise Error
```

Specifically, if we have a contract where we have to pay for searches, then exercising the search right will create a payment obligation correspondingly (where the exact amount might depend on parameters of the search again).

4.5 Client Example: Other Programs ("Agents") Interfacing with FIRM Objects

One of the often-quoted scenarios in electronic commerce is the ability of having an "agent" roaming around the network and trying to find the cheapest offer from a variety of sources. Let us outline here how client programs can interface with FIRM objects in a way that enables such usages in a very general way, that is, in a way that not only can take basic attributes such as the price of a good into account, but in principle also any other types of terms and conditions (such as policies about return-of-goods, etc.).

Given a contract offer with a certain payment obligation, any program can ask for the attribute model of this contract. The following code fragment shows how a program can find out the price of an offer. Note that in this case, the price is a per-item price—a fact that will be known to the programmer of the agent and therefore can be considered in comparing this per-item price with other prices that might be using a different pricing model.

```
payOblig = offer.FindPromise("Payment")
theModel = payOblig.GetAttributeModel()
if theModel.Name()=="SimplePayPerUsePaymentObligationModel":
  attribNameForPrice = "pricePerUse"
...
priceInThisOffer = payOblig[attribNameForPrice]
```

4.6 Interoperability Example: Unix File Rights into the FIRM Object Model

Let us consider here how FIRM could provide an interoperability "wrapper" for the basic access rights that the Unix file system provides. We could have such wrappers for the various file servers that people with different accounts use (next to corresponding wrappers for Windows NT, for instance). This would allow us to use the FIRM service layer to determine and manipulate access rights in a uniform way.

Specifically, let us consider here the Unix 'read' right as an example. We would like to have an implementation for a FIRM rights object that is a wrapper to the Unix read right

for a specific file. This implementation could be just a proxy to the real Unix file access control in the following way (although there clearly could be other conceptualizations and implementations).

We assume that the generic promise behavior has already been implemented by `PromiseImpl.CPromise`; in particular, this right then already knows who it is held by, which file it is about, etc. Moreover, we assume that there is some commpact that encapsulates the relationship that a specific person has with the provider of the Unix account on a certain system. This might include disk space quotas, etc. In particular, it provides the necessary information to deal with the relevant authentication issues involved. The account names are made available via an attribute model in this case that has an attribute “UnixUserIDs.”. This provides the required mapping from the identity of an e-person to the account name that this person has with a particular Unix system. In other words, we mostly need to provide an implementation of the ‘exercise’ methods:

```
class UnixReadRight(PromiseImpl.CPromise):
  def ShortDescription(self):
    page = PromiseImpl.CPromise.ShortDescription(self)
    page = page + “The holder of this right has unlimited
                    read access to the files.“
    return page

  # An auxiliary method that gets the Unix account name
  # that corresponds to the right holder’s e-person handle.
  def convertEpersID2UnixAccountID(epersRef):
    return self.cpctContext[“UnixUserIDs”][self.Holder()]

  def Exercise(self, actor, inParams):
    # Check whether actor is authorized to exercise this right;
    # and do other general checks.
    PromiseImpl.CPromise.Exercise(actor, inParams)
    # Get local user name
    try:
      epersUID = self.convertEpersID2UnixAccountID(self.Holder())
    except:
      raise FIRM.NotAuthorized, “Do not have a local account“
    # Get the file that this read request is about
    fileName = inParams[“requestedFile“]
    # Now proxy to Unix access read access right:
    # Get the Unix file descriptors for this file.
     (st_mode, st_ino, st_dev, st_nlink, st_uid, st_gid, st_size,
      st_actime, st_mtime, st_ctime) = posix.stat(fileName)
    # Check whether user has read right
    if epersUID<>st_uid:
      raise FIRM.NotAuthorized, “Not the file’s Unix owner“
    ownerReadRight = string.atoi(oct(st_mode)[-3])
    if not (ownerReadRight % 2):
      raise FIRM.NotAuthorized, “Do not have read access“
    # If we come to this point, then we know that we have
    # read access to the file according to Unix.
    # At this point, we could perform additional state keeping
    # functions, such as cacheing, logging accesses, etc.
```

References

1.0 Privacy, Copyright, Intellectual Property, Legal and Economic Aspects

1.1 Legal Aspects

1.1.1 Contract Law

[1] Gilbert Law Summaries. (1985). *Contracts*. By Eisenberg, M.A. Harcourt, Brace, Jovaovich Legal and Professional Publications.

[2] Craswell, R., and A. Schwatz (1994). *Foundation of Contract Law.* Oxford University Press.

[3] Atiyah, P.S. (1995). *An Introduction to the Law of Contract.* Clarendon Law Series. Fifth Edition. Clarendon Press, Oxford.

[4] MacNeil, I.R. (1985). Relational Contract: What We Do and Do Not Know. *Wisconsin Law Review* 483.

[5] Whitford, W.C. (1985). Ian MacNeil's Contribution to Contracts Scholarship. *Wisconsin Law Review* 545.

[6] MacNeil, I.R. (1974). The Many Futures of Contracts. *Southern California Law Review* **47**(691).

[7] Barnett, R.E. (1986). A Consent-Theory of Contract. *Columbia Law Review* **86**(269).

[8] Barnett, R.E. (1992). Conflicting Visions: A Critique of Ian MacNeil's Relational Theory of Contract. *Virginia Law Review* **78**(1175).

[9] Fried, C. (1981). Contract as Promie: A Theory of Contractual Obligation. Harvard University Press.

[10] Linzer, P. (1995, eds.). A Contracts Anthology. Anderson Publishing. pp. 54-128.

1.1.2 Electronic Contracting

[11] Baum, M. (1989). Electronic Contracting in the U.S.: The Legal and Control Context. In I. Walden (ed.), *EDI and the Law.* Blenheim Online, London.

[12] Greguras, F.M., T.A. Golobic, R.A. Mesa, R. Duncan (1995). On-line Contract Issues. Updated version of a presentation made at *Law Seminars International Electronic Commerce: Doing Business On-line*, September 21, 1995. Web: http://www.batnet.com/oikoumene/ec_contracts.html.

[13] Allen, T., and R. Widdison (1996). Can Computers Make Contracts? *Harvard Journal of Law & Technology* **9**(25), Harvard Law School.

[14] Wright, B. (1995). *The Law of Electronic Commerce. EDI, E-Mail, and Internet: Technology, Proof, and Liability.* Little, Brown and Co.

1.1.3 Copyright

[15] Goldstein, P. (1994). *Copyright's Highway: The Law and Lore of Copyright from Gutenberg to the Celestial Jukebox.* Hill and Wang.

[16] Greguras, F. (1995). Copyright Clearances and Moral Rights. *Softic Symposium '95*, November 1995.

[17] Canadian Copyright Subcommittee (1995). *Report on Copyright and the Information Highway.* Information Highway Advisory Council. September. Available at http://debra.dgbt.doc.ca/info-highway/ih.html.

[18] U.S. Government (1995). *Intellectual Property and the National Information Infrastructure.* Report of the Working Group on Intellectual Property Rights. Lehman, Bruce, and Ronald Brown, Information Infrastructure Task Force. September. Washington, DC.

1.1.4 Property

[19] Gilbert Law Summaries (1990). *Property.* By Dukeminier. M. Harcourt, Brace, Jovaovich Legal and Professional Publications.

[20] Rose, C.M. (1994). *Property and Persuasion: Essays on the History, Theory, and Rhetoric of Ownership*. Westview Press, Inc.

[21] Radin, M.J. (1993). *Reinterpreting Property.* University of Chicago Press.

[22] Branscomb, A.W. (1994). *Who Owns Information ? From Privacy to Public Access*. New York: Basic Books.

[23] Perritt, H. (1994). Permission Headers and Contract Law. *Proceedings of the Workshop on Technological Strategies for Protecting Intellectual Property in the Networked Multimedia Environment*. Coalition for Networked Information, Interactive Multimedia Association, and John F. Kennedy School of Government.

[24] Jensen, M. (1994). Need-Based Intellectual Property Protection and Networked University Press Publishing. *Proceedings of the Workshop on Technological Strategies for Protecting Intellectual Property in the Networked Multimedia Environment*. Coalition for Networked Information, Interactive Multimedia Association, and John F. Kennedy School of Government.

1.1.5 Other

[25] Rose, Lance (1995). *NewLaw: Your Rights in the Online World*. Osborne McGraw-Hill, Berkeley.

[26] Nimmer, R., and P. Krauthaus (1992). Information as a Commodity: New Imperatives of Commercial Law. *Law and Contemporary Problems* **55**(3).

[27] Reidenberg, Joel (1993). Rules of the Road for Global Electronic Highways: Merging the Trade and Technical Paradigms. *Harvard Journal of Law & Technology* 287, 289.

1.2 Business Aspects

[28] Dyson, Esther (1995). Intellectual Value. *Wired*, Issue 3.07. Excerpt from December 1994 issue of *Release 1.0*, EDventure Holdings.

[29] McKenna, Regis (1991). *Relationship Marketing: Successful Strategies for the Age of the Customer.* Addison-Wesley.

[30] McKenna, Regis (1997). *Real-Time: Preparing for the Age of the Never Satisfied Customer.* Harvard Business School Press.

[31] Peppers, Don, and Martha Rogers (1993). *The One to One Future: Building Relationships One Customer At a Time*. Currency Doubleday.

[32] Mansfield, Nick (1996). Security at Shell Int'l. *Proceedings of the Sixth Conference on Computers, Freedom, and Privacy*, Boston. Presentation by Nick Mansfield, Information Security Advisor for Shell Companies. Web: http://swissnet.ai.mit.edu/~switz/cfp96/plenary-crypto.html

[33] National Writers Union (1994). Statement of Principles on Contracts between Writers and Electronic Book Publishers. Available at ftp://ftp.netcom.com/pub/nw/nwu/press/online-pub.txt.

[34] Moss, N. (1996). Europe's slow-motion view: Hollywood is fighting to retain control of film release dates. Report by Nicholas Moss, *The European*, June 3rd, 1996.

1.3 Economic Aspects

[35] Bressand, Albert, and Catherine Distler (1995). *La plan'ete relationnelle*. Paris: Flammarion.

[36] Coase, R.H. (1988). *The Firm, the Market, and the Law.* University of Chicago Press, London.

[37] Coase, R.H. (1990). *Institutions, Institutional Change and Economic Performance.* Cambridge University Press.

[38] Greif, Avner (1994). Cultural Beliefs and the Organization of Society: Historical and Theoretical Reflection on Collectivist and Individualist Societies. *The Journal of Political Economy*, October.

[39] Greif, Avner (1992). Institutions and Commitment in International Trade: Lessons from the Commercial Revolution. *American Economic Review* **82**(5), pp. 128-133.

[40] Greif, A., P. Milgrom, and B. Weingast (1992). The Merchant Guild as a Nexus of Contracts. Mimeo, Stanford University.

[41] Milgrom, Paul, and John Roberts (1992). *Economics, Organization, and Management*. Prentice Hall, NJ.

[42] Williamson, O. (1985). *The Economic Institutions of Capitalism.* The Free Press, NY.

[43] Williamson, O. (1975). *Markets and Hierarchies: Analysis and Antitrust Implications.* The Free Press, NY.

[44] Williamson, O. (1986). *Economic Organization: Firms, Markets and Policy Control*. New York University Press, NY.

[45] North, Douglas C. (1990). *Institutions, Institutional Change and Economic Performance*. Cambridge University Press.

[46] Hardwick, M., D.L. Spooner, T. Rando, and K.C. Morris (1996). Sharing manufacturing information in virtual enterprises. *Communications of the ACM* **39**(2):46-53.

1.4 Enforcement of Informal Constraints

[47] Ellickson, R. (1986). Of Coase and Cattle: Dispute Resolution Among Neighbors in Shasta County. *Stanford Law Review,* 38:624-87.

[48] Ellickson, R. (199?). *Order without Law.* Harvard University Press, Cambridge, MA.

[49] Bentham, Jeremy (1787). *Panopticon; or, The inspection-house: containing the idea of a new principle of construction applicable to any sort of establishment, in which persons of any description are to be kept under inspection: and in particular to penitentiary-houses, prisons, houses of industry ... and schools: with a plan of management adapted to the principle.* Dublin printed; London, Re-printed and sold by T. Payne, 1791.

[50] Semple, Janet (1993). *Bentham's Prison: A Study of the Panopticon Penitentiary.* Oxford University Press.

[51] Williams, Monte (1996). Sex offenders law prompts privacy debate in New York. *The New York Times*, A1, January 24.

1.5 Privacy, Personal Information

1.5.1 General

[52] Warren, Samuel, and Louis Brandeis (1890). The Right to Privacy. *Harvard Law Review* **193**.

[53] Agre, Phil (1994). Surveillance and Capture: Two Models of Privacy. *The Information Society* **10**(2), pp. 101-127.

[54] Burns, R., R. Samarajiva, and R. Mukherjee (1992). *Customer Information: Competitive and Privacy Implications*. Columbus, OH: National Regulatory Institute.

[55] Gandy, O.H., Jr. (1993). *The Panoptic Sort: A Political Economy of Personal Information.* Boulder CO: Westview.

[56] Goffman, E. (1971). *Relations in Public: Microstudies of the Public Order.* New York: Basic Books.

[57] Goffman, E. (1963). *Behavior in Public Places.* The Free Press.

[58] Gottdiener, M. (1985). *The Social Production of Urban Space*. Austin, TX: University of Texas Press.

[59] Jussawalla, M., and C. Chee-Wah (1987). Economic Analysis of the Legal and Policy Aspects of Information Privacy. Chapter 4 in *The Calculus of International Communications.* Littleton, CO.

[60] Karnow, Curtis E.A. (1994). The Encrypted Self: Fleshing out the Rights of Electronic Personalities. *Conference* on *Computers, Freedom, and Privacy.*

[61] Miller, A.R. (1969). Personal Privacy in the Computer Age: The Challenge of New Technology in an Information-Oriented Society. *Michigan Law Review* **67**: 1224-25.

[62] Samarajiva, R. (1994). Electronic Public Space: Dystopic and other Futures. *Computers, Freedom, and Privacy Conference*.

[63] Stanley, T. (1994). Electronic Communications Privacy Rights. CPSR Civil Liberties Project. URL: http://www-leland.stanford.edu/~tstanley/cpsrart.html.

[64] Turn, R. (1990). Information Privacy Issues for the 1990s. *IEEE Computer Society Symposium on Research in Security and Privacy.*

[65] Arms, B. (1994). Key Concepts in the Architecture of the Digital Library. *D-Lib Journal*, July. Web: http://www.cnri.reston.va.us/home/dlib/July95/.

[66] Bellotti, V., and A. Sellen (1993). Design for Privacy in Ubiquitious Computing Environments. *Proceedings of the Third European Conference on Computer-Supported Cooperative Work,* Milan, Italy.

[67] Chaum, D. (1992). Achieving Electronic Privacy. *Scientific American*, August.

[68] Rotenberg, M. (1993). Communications Privacy: Implications for Network Design. *Communications of the ACM* **36**(8), pp. 61-68.

1.5.2 Studies and Guidelines

[69] OECD (1980). *Guidelines Governing the Protection of Privacy and Transborder Flows of Personal Data*. Annex to Recommendations of the Council of 23rd September 1980, Organization for Economic Cooperation and Development.

[70] Lawson, Ph., and M. Vallee (1995). Canadians Take Their Information "Personal." *Privacy Files* **1**(1), pp. 4-9, October. Progesta Publishing, Canada.

[71] Ekos Research Associates (1993). *Privacy Revealed: The Canadian Privacy Survey.* Ekos. Ottawa, Cananda.

[72] Canadian Standards Association (1995). *CSA Model Code for Protection of Personal Information*. CAN/CSA-Q830-1995, August. Working Draft.

[73] European Union (1995). *Directive of the European Parliament and of the Council on the Protection of Individuals with Regard to the Processing of Personal Data and on the Free Movement of Such Data*. July 20th, Brussels.

[74] U.S. Congress (1995). *Information Security and Privacy in Network Environments.* Office of Technology Assessment Study. Washington, DC.

[75] U.S. Government (1995). *Privacy and the National Information Infrastructure: Principles for Providing and Using Personal Information.* Information Infrastructure Task Force, Information Policy Committee, Privacy Working Group. Washington, DC.

[76] U.S. Congress (1991). *Domestic and International Data Protection Issues*. Hearings before the Government Information, Justice, and Agriculture Subcommittee of the Committee on Government Operations, April 10, 1991. Washington, DC.

[77] Equifax (1990). *The Equifax Report on Consumers in the Information Age*. Survey conducted by Louis Harries & Associates and Dr. Alan Westin. Reprinted in Congressional Hearings [76], pp. 290-435.

[78] Westin, Alan F. (1991). How the American public views consumer privacy issues in the early 90's and why. Testimony before the Subcommittee on Government Information, Justice, andAgriculture. House Committee on Government Operations, Washington, D.C., April 10, 1991.

[79] Equifax (1995). *The 1995 Equifax-Harris Mid-Decade Consumer Privacy Survey.* Survey conducted by Louis Harries & Associates and Dr. Alan Westin. Available from URL http://www.equifax.com/.

[80] Westin, Alan F. (1991). Interpretive Essay. Interpretation of the Findings in Equifax [79]. Web: http://www.equifax.com/.

[81] Garfinkel, Simon (1995). Separating Equifax from Fiction. *Wired* 3.09, September, pp. 96-107.

1.5.3 Medical Information

[82] Westin, Alan F. (1976). *Computers, Health Records, and Citizen's Rights*. U.S. Department of Commerce.

[83] Medical Records (1996). Report on the US Medical Records Confidentiality Act, sponsored by Senator Robert Bennett and Patrick Leary. *The New York Times*, Nov 15, 1995, A1 ("Medical Records Are on Sale in the Marketplace").

[84] Privacy of Medical Records (1979). Hearings before a Subcommittee of the House Committee on Government Operations, 96th Cong., 1st Sess.

[85] Legislation to Protect the Privacy of Medical Records (1979). Hearings before the Senate Committee on Governmental Affairs, 96th Cong. 1st Sess.

[86] House Committee on Government Operations (1980). Federal Privacy of Medical Information Act, H.R. Rep. No 96-832 Part 1, 96th Cong., 2d Sess.

[87] Data Protection, Computers, and Changing Information Practices (1990). Hearing before the Subcomm. on Government Information, Justice, and Agriculture, House Comm. on Government Operations, 101st Cong., 2d Sess.

[88] Health Reform, Health Records, Computers and Confidentiality (1993). Hearing before the Information, Justice, Transportation, and Agriculture Subcomm. of the House Committee on Government Operations, 103rd Cong., 1st Sess.

[89] Fair Health Information Practices Act of 1994 (1994). Hearings before the Information, Justice, Transportation, and Agriculture Subcomm. of the House Committee on Government Operations, 103rd Cong., 2d Sess.

[90] House Committee on Government Operations (1994). Health Security Act, H.R. Rep. No 103-601 Part 5, 103rd Cong., 2d Sess.

[91] Rindfleisch, T. (1997). Privacy and Security in Health Care. *Communications of the ACM*, August.

2.0 Access Control, Rights Management

2.1 General

[92] Saltzer, J.D., and M.D. Schroeder (1975). The Protection of Information in Computer Systems. *Proceedings of the IEEE* **63**(9), pp. 1278-1308.

[93] ERMG (1995). Minutes of the first meeting of the Electronic Rights Management Group (Boston, MA), Information Industries Association, October 31, 1995.

[94] ISO (1989). Security Framework III: Access Control Framework. ISO/IEC JTC1/SC21 N4206. Draft, November.

[95] Silberschatz, A., J. Peterson, and P. G. Galvin (1991). *Operating Systems Concepts*. Addison-Wesley.

[96] Weber, Robert (1995). Digital Rights Management Technologies. I*nternational Federation of Reproduction Rights Organization*, Danvers, MA. October 1995.

[97] Stefik, M. (1995). Letting loose the light: Igniting commerce in electronic publishing. Draft, Xerox Palo Alto Research Center, Palo Alto, CA.

[98] Cyberspace Law Center (1997). Accessible at http://www.cybersquirrel.com/clc/

[99] Stefik, M. (1996). Digital Property Rights: Technology, Choices, and Social Values. Xerox Palo Alto Research Center, Palo Alto, CA.

2.2 Conceptual Models

[100] Lampson, B.W. (1971). Protection. *5th Princeton Symposium on Information Science and Systems*. Reprinted in *ACM Operating Systems Review* **8**(1):18-24, 1974.

[101] Harrison, M.H., W.L. Ruzzo, and J.D. Ullman (1976). Protection in operating systems. *Communications of the ACM* **19**(8), pp. 461-471.

[102] Marc, D. (1993). A Petri Net Representation of the Take-Grant Model. *Proceedings of the IEEE Symposium on Security and Privacy.*

[103] Minsky, N. (1977). Cooperative authorization in computer systems. *IEEE Computer Society's First International Computer Software and Applications Conference*, pp.729-33.

[104] Minsky, N. (1978). An operation-control scheme for authorization in computer systems. *International Journal of Computer & Information Sciences* **7**(2), pp. 157-91.

[105] Minsky, N.H., and A.D. Lockman (1985). Ensuring integrity by adding obligations to privileges. *Proceedings of the 8th International Conference on Software Engineering*, pp. 92-102.

[106] Sandhu, R.S. (1989). Transformation of Access Rights. *Proceedings of the IEEE Symposium on Security and Privacy.*Oakland, CA.

[107] Sandhu, R.S., and G.S. Suri (1992). Non-Monotonic Transformation of Access Rights. *Proceedings of the IEEE Symposium on Security and Privacy.*Oakland, CA.

[108] Sandhu, R.S. (1992). The Typed Access Matric Model. *Proceedings of the IEEE Symposium on Security and Privacy.*Oakland, CA.

[109] Sandhu, R.S. (1988). The Schematic Protection Model: Its Definitions and Analysis for Acyclic Attenuating Schemes. *Journal of ACM* **8**(2):404-432.

[110] Sandhu, R.S., and M.E. Share (1986). Some Owner-based Schemes with Dynamic Groups in the Schematic Protection Model. *Proceedings of the IEEE Symposium on Security and Privacy.*

[111] Thomas, R.K., and R.S. Sandhu (1993). Towards a Task-based Paradigm for Flexible and Adaptable Access Control in Distributed Applications. *Proceedings of the Second New Security Paradigms Workshop.* Little Compton, Rhode Island.

[112] Thomas, R.K., and R.S. Sandhu (1994). Conceptual Foundations for a Model of Task-based Authorizations. *Proceedings of the IEEE Conference on Security and Privacy.*

[113] Moffett, J.D., and M. S. Sloman (1991). Content-dependent Access Control. *Operating Systems Review* **25** (2), pp. 63-70, April.

[114] Strack, H., and K. Lam (1993). Context-dependent Access Control in Distributed Systems. *IFIP Transactions in Computer Security* A-37. Elsevier Publishers.

[115] Abrams, M.D., and M.V. Joyce (1993). Extending the ISO Access Control Framework for Multiple Policies. *IFIP Transactions in Computer Security* A-37. Elsevier Publishers.

[116] Abrams, M.D., and I.M. Olsen (1992). Rule-based Trusted Access Control. In G.G. Gable and W.J. Caelli (eds.), *IT Security: The Need for International Cooperation*. Elsevier Publishers.

2.3 Authorization Languages

[117] Stefik, M. (1996). The Digital Property Rights Language. Manual and Tutorial. Version 1.02, September 18th. Xerox Palo Alto Research Center, Palo Alto.

[118] Upthegrove, Luella, and T. Roberts (1994). Intellectual Property Header Descriptors: A Dynamic Approach. *Proceedings of the Workshop on Technological Strategies for Protecting Intellectual Property in the Networked Multimedia Environment*. Coalition for Networked Information, Interactive Multimedia Association, and John F. Kennedy School of Government.

[119] CODASYL Data Description Language Committee (1987). Report. *Information Systems* **3**(4), pp. 247-320.

[120] W3O (1994). WWW Access Authorization. URL: http://www.w3.org/hypertext/WWW/AccessAuthorization/Overview.html.

[121] Koster, M. (1994). A Standard for Robot Exclusion. Web: http://info.webcrawler.com/mak/projects/robots/norobots.html.

[122] Morris, J. H. (1973). Protection in Programming Languages. *Communications of the ACM* **16**(1). pp. 15-21.

[123] Kieburtz, R. B., and A. Silberschatz (1983). Access Right Expressions. *ACM Transactions on Programming Languages and Systems* **5**(1), pp. 78-96.

[124] Abadi, M., M. Burrows, and B. Lampson (1993). A Calculus for Access Control in Distributed Systems. *ACM TOPLS* **15**(4), pp. 706-734, September.

[125] Hoffmann, L.J. (1971). The Formulary Model for Flexible Privacy and Access Control. *AFIPS Conf. Proc*. **39**, FJCC, 587-601. AFIPS Press, Montvale, NJ.

[126] La Padula, L. (1990). Formal Modeling in a Generalized Framework for Access Control. *Proceedings of the IEEE Symposium on Security and Privacy,*Oakland, CA.

[127] Chrysanthis, P.K., and K. Ramamritham (1990). ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behavior. *Proceedings of the ACM SIGMOD conference*, pages 194-203.

[128] Sandhu, R.S. (1988). Transaction Control Expressions for Separation of Duties. *Proceedings of the Fourth Computer Security Applications Conference*, pp. 282-286.

[129] Woo, Th., and S. Lam (1992). Authorization in Distributed Systems: A Formal Approach. *Proceedings of the IEEE Conference on Security and Privacy*, Oakland, CA.

2.4 Implementation Models

[130] Hauser, R. (1993). Does Licensing Require New Access Control Techniques ? *Proceedings of the First ACM Conference on Computer and Communications Security*, pp. 1-8. Fairfax, VA.

[131] Kahan, J. (1994). Un Nouveau Modèle d'Autorisation pour les Systèmes de Consultation d'Information Multimédia Repartie. *AFCET*, Télécom Paris.

[132] Zurko, M.E. (1992). Attribute Support for Inter-Domain Use. *Proceedings of the IEEE Conference on Security and Privacy,* Oakland, CA.

2.5 Revocation

[133] Redell, D. (1974). *Naming and Protection in Extendible Operating Systems*. PhD dissertation, UC Berkeley. Also: MIT MAC TR TR-140.

[134] Ekanadham, K., and A. J. Bernstein (1979). Conditional Capabilities. *IEEE Transactions on Software Engineering* SE-**5**(5), pp. 458-464.

2.6 Systems

[135] Blaze, M. et al. (1996). PolicyMaker. Web: ftp://research.att.com/dist/mab/.

[136] Blaze, M., J. Feigenbaum, and J. Lacy (1996). Decentralized Trust Management. *IEEE Symposium on Security and Privacy*, Oakland CA, May 1996.

[137] Corbato, F. J., and V. A. Vyssotsky (1965). Introduction and Overview of the MULTICS System. *Proceedings of AFIPS SJCC*, pp. 185-196.

[138] Levin, R., E. S. Cohen, W. M. Corwin, F. J. Pollack, and W. A. Wulf (1975). Policy/Mechanism Separation in Hydra. *Proceedings of the Fifth ACM Symposium on Operating System Principles*, pp. 132-140.

[139] Cohen, E. S., and D. Jefferson (1975). Protection in the Hydra Operating System. *Proceedings of the Fifth ACM Symposium on Operating System Principles*, pp. 141-160.

[140] Needham, R. M., and R. D. H. Walker (1977). The Cambridge CAP Computer and its Protection System. *Proceedings of the Sixth ACM Symposium on Operating System Principles*, pp. 1-10.

[141] Cox, B., J.D. Tygar, and M. Sirbu (1995). NetBill Security and Transaction Protocol. Technical Report, Carnegie-Mellon University.

[142] Chu, Y.-H., J. Feigenbaum, B. LaMacchia, P. Resnick, M. Strauss (1997). Referee: Trust Management for Web Applications. 6th Web Conference, Stanford.

[143] Gladney, H,M. (1996). Digital Intellectual Property—Risks and Protection Mechanisms. IBM Almaden Reseach Center, San Jose, CA.

[144] Gladney, H.M. (1992). Access Control for Large Collections. IBM Research Report RJ 8946. Also in *ACM Trans. Information Systems*.

[145] InterTrust (1995). InterTrust Electronic Rights System. InterTrust, Incorporated. URL: http://www.intertrust.com (formerly Electronic Publishing Resources, http://www.epr.com)

[146] Erickson, J.S. (1994). Electronic Copyright Management in the Production of Networked Interactive Multimedia. PhD thesis proposal, Thayer School of Engineering, Dartmouth College.

[147] Rivest, R., and B. Lampson (1996). SDSI—A Simple Distributed Security Infrastructure. Technical Report. Massachusetts Institute of Technology, Cambride, MA.

[148] Rotenberg, L. (1973). Making computers keep secrets. PhD dissertation, MIT, Cambridge, Mass. Also: MIT MAC TR-115.

[149] Ritchie, D. M., and K. Thompson (1978). The UNIX Time-Sharing System. *Communications of the ACM* **17**(7), pp. 365-375, July.

[150] Kahn, R.E. (1994). Deposit, Registration, and Recordation in an Electronic Copyright Management System. *Proceedings of the Workshop on Technological Strategies for Protecting Intellectual Property in the Networked Multimedia Environment*. Coalition for Networked Information, Interactive Multimedia Association, and John F. Kennedy School of Government.

[151] Library of Congress (1996). The CORDS copyright management system. Web: http://lcweb.loc.gov/copyright/cords.html

2.7 Watermarking

[152] Low, Maxemchuk, Brassil, O'Gorman (1993). *Document Marking and Identification using both Line and Word Shifting.* Web: ftp://ftp.research.att.com/dist/brassil/docmark2.ps.

[153] Choudhury, Maxemchuk, Paul, Schulzrinne (1994). *Copyright Protection for Electronic Publishing over Computer Networks.* Web: ftp://ftp.research.att.com/dist/anoncc/copyright.epub.ps.Z.

[154] Brassil, Low, Maxemchuk & O'Gorman (1994). *Electronic Marking and Identification Techniques to Discourage Document Copying.* Web: ftp://ftp.research.att.com/dist/brassil/infocom94.ps.

3.0 Electronic Contracting, EDI

[155] EDI (1979). ANSI ASC X12. ISO/IEC JTC1/SWG-EDI.

[156] EDIFACT (1995). UN/EDIFACT Standards (EDI for Adminstration, Commerce, and Transport). Web: http://www.premenos.com/unedifact/.

[157] UN/ECE (1994). UN/EDIFACT Message Design Guidelines. Web: http://www.premenos.com/standards/.

[158] UN/ECE (1994). General Introduction to UNSM Descriptions. Web: http://www.premenos.com/standards/.

[159] Open-EDI (1994). Open-EDI Conceptual Model. ISO/IEC JTC1/SWG-EDI N222.

[160] Hill, N., and D. Ferguson (1995). Electronic Data Exchange: A Definition and Perspective. *EDI Aware*, Issue 4, Winter. Web: http://infopole1.soca.cf.ac.uk/edi/EDIAware4Index.html.

[161] Nelson, C. (1995). The ABC of EDI. *EDI Aware*, Issue 4, Winter. Web: http://infopole1.soca.cf.ac.uk/edi/EDIAware4Index.html.

[162] UN/ECE (1995). Electronic Data Interchange Standards. Web: http://www.premenos.com/standards/.

4.0 Security, Integrity, and Cryptography

4.1 General

[163] Anderson, R. (1995). Computer and Communications Security Reviews. Web: http://www.cl.cam.ac.uk/users/rja14/.

[164] Amoroso, E. (1994). *Fundamentals of Computer Security Technology.* Prentice Hall. Englewood Cliffs, NJ.

[165] Fernandez, E., R. Summers, and C. Wood (1981). *Database Security and Integrity.* Addison-Wesley.

[166] Kaufman, C., R. Perlman, and M. Speciner (1995). *Network Security: Private Communications in a Public World*. Prentice Hall, NJ.

[167] Anderson, J. P. (1972). Computer Security Technology Planning Study. ESD-TR-73-51, AD-758 206, ESD/AFSC Hanscom.

[168] Anderson, R.J. (1993). Why Cryptosystems Fail. *Proceedings of the First ACM Conference on Computer and Communications Security*, Fairfax, pp. 215-227.

[169] Lampson, B.W. (1973). A Note on the Confinement Problem. *Communications of the ACM* **16**(10), pp. 613-615.

[170] Smith, M. (1994). A People Problem. *International Security Review* **83**.

[171] Schneier, B. (1994). *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley and Sons, NY.

4.2 Authentication

[172] Birrel, A., B. Lampson, R. Needham, and M. Schroeder (1986). A Global Authentication Service without Global Trust. *Proceedings of the IEEE Symposium on Security and Privacy.*

[173] Wobber, E., M. Abadi, M. Burrows, and B. Lampson (1993). Authentication in the Taos Operating System. *ACM SIGOPS*, pp. 256-269.

[174] Dennis, J., and E. van Horn (1966). Programming Semantics for Multi-programmed Computations. *Communications of the ACM* **9**(3), pp. 143-155.

[175] Yahalom, R., B. Klein, and Th. Beth (1993). Trust Relationship in Secure Systems—A Distributed Authentication Perspective. *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA.

4.3 Security and Integrity

[176] Denning, D. (1993). A New Paradigm for Trusted Systems. *Proceedings of the New Security Paradigms Workshop*. Little Compton, Rhode Island.

[177] Department of Defense (1985). Trusted Computer Systems Evaluation Criteria. DOD 5200.28-STD, December.

[178] Dobson, John (1993). New Security Paradigms: What Other Concepts Do We Need as Well? *Proceedings of the New Security Paradigms Workshop.* Little Compton, Rhode Island.

[179] LaPadula, L.J., and J.G. Williams (1991). Towards a Universal Integrity Model. *Proceedings of the IEEE Computer Security Foundations Workshop.* Franconia, New Hampshire. IEEE Press.

[180] McCullough, D. (1987). Specification for Multi-level Security and a Hook-up Property. *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA.

[181] Sandhu, R.S. (1990). On the Five Definitions of Data Integrity. *Proceedings of the 7th Annual IFIP Working Conference on Database Security.* Huntsville, Alabama.

[182] Minsky, N., and V. Ungureano (1997). Unified Support for Heterogeneous Security Policies in Distributed Systems. Department of Computer Science, Rutgers University.

[183] Minsky, N., and V. Ungureano (1997). A Framework for Supporting Heterogeneous Coordination Policies. Department of Computer Science, Rutgers University.

[184] Wiederhold, G., M. Bilello, V. Sarathy, and X. Qian (1996). A Security Mediator for Health Care Information. *Proceedings of the 1996 AMIA (formely SCAMC) Conference*, Oct. 1996, pp.120-124.

[185] Wiederhold, G., Michel B., V. Sarathy, and X. Qian (1996). Protecting Collaboration. *Proceedings of the NISSC'96 National Information Systems Security Conference*, pp. Oct. 1996, pp.561-569.

[186] Qian, X., G. Wiederhold, M. Bilello, A. Chavez, and V. Sarathy (1996). Trusted Interoperation of Healthcare Information. *Abstract for the NSF Challenge workshop*, Stanford, March 20-23, 1996.

4.4 Specific Security Policies and Models

[187] Bell, D.E., and L.J. LaPadula (1976). Secure Computer Systems: Unified Exposition and Multics Interpretation, Report No. MTR-2997, MITRE, Bedford, MA.

[188] Biba (1977). Integrity Considerations for Secure Computer Systems. ESD-TR-76-372, USAF Electronic Systems Division, Bedford, MA.

[189] Brewer, D., and M. Nash (1989). The Chinese Wall Security Policy. *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA.

[190] Clark, D.D., and D.R. Wilson (1987). A Comparison of Commercial and Military Security Policies. *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA.

[191] Dobson, J.E., and J.A.McDermid (1989). Security Models and Enterprise Models. In C.E. Landwehr (ed.), *Database Security, II: Status and Prospects*, Elsevier Science Publishers, Amsterdam.

[192] Goguen, J.A., and J. Meseguer (1982). Security Policy and Security Models. *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA.

[193] McLean, J. (1990). Security Models and Information Flow. *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA.

[194] Sandhu, R.S. (1990). Separation of Duties in Computerized Information Systems. In C.E. Landwehr (ed.), *Database Security, IV: Status and Prospects*, Elsevier Science Publishers, Amsterdam.

[195] Sterne, D. (1991). On the Buzzword "Security Policy". *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA.

[196] Sutherland, D. (1986). A Model of Information. *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA.

[197] Williams, J. (1993). A Shift in Security Modeling Paradigms. *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA.

4.5 Cryptography

[198] Hoffman, L., F. Ali, S. Heckler, and A. Huybrechts (1993). Cryptography: Policy and Technology Trends. *Conference on Computers, Freedom, and Privacy.*

[199] Diffie, W., and M. E. Hellman (1976). New Directions in Cryptography. *IEEE Transactions on Information Theory* **22**(6), pp. 397-427.

[200] Rivest, R. L., A. Shamir, and L. Adleman (1978). On Digital Signatures and Public Key Cryptosystems. *Communications of the ACM* **21**(2), pp. 120-126.

[201] Chaum, D. (1985). Showing Credentials without Identification: Signatures transferred between Unconditionally Unlinkable Pseudonyms. In *Advances in Cryptology—Eurocrypt '85*, pp. 241-244. Springer Verlag.

5.0 Standards Process Documents

5.1 IETF Standards Track RFCs

[202] Linn, J. (1993). *Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures*. RFC1421. Web: http://ds.internic.net/rfc/.

[203] Kent, S. (1993). *Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management.* RFC1422. Web: http://ds.internic.net/rfc/.

[204] Kalinski, B. (1993). *PEM IV: Key Certification and Related Services.* RFC1424. Web: http://ds.internic.net/rfc/.

[205] Kaufman, C. (1993). *DASS - Distributed Authentication Security Service.* RFC1507. Web: http://ds.internic.net/rfc/.

[206] Linn, J. (1993). *Generic Security Service Application Program Interface.* RFC1508. Web: http://ds.internic.net/rfc/.

[207] Kohl, J., and C. Neuman (1993). *The Kerberos Network Authentication Service (V5).* RFC1510. Web: http://ds.internic.net/rfc/.

5.2 IETF Informational RFCs

[208] Braden, R., D. Clark, S. Crocker, C. Huitema (1994). *Report of IAB Workshop on Security in the Internet 1994.* RFC1636. Web: http://ds.internic.net/rfc/.

[209] Haller, N., and R. Atkinson (1994). *On Internet Authentication.* RFC1704. Web: http://ds.internic.net/rfc/.

5.3 CCITT Standards

[210] CCITT X.509 (1988). Recommendation X.509: The Directory—Authentication Framework. Consultative Committee on International Telegraphy and Telephony.

[211] CCITT X.500 (1988). Recommendation X.500: The Directory—Overview of Concepts, Models and Services. Consultative Committee on International Telegraphy and Telephony.

[212] CCITT X.501 (1988). Recommendation X.501: The Directory—Models. Consultative Committee on International Telegraphy and Telephony

[213] CCITT X.208 (1988). Recommendation X.208. Abstract Syntax Notation 1 (ASN.1). Consultative Committee on International Telegraphy and Telephony.

[214] CCITT X.209 (1988). Recommendation X.209. Basic Encoding for ASN.1. Consultative Committee on International Telegraphy and Telephony.

5.4 RSA Laboratories Standards

[215] RSA Laboratories (1993). *PKCS #6: Extended-Certificate Syntax Standard.* Web: ftp://ftp.rsa.com/pub/pkcs/.

[216] RSA Laboratories (1993). *PKCS #9: Selected Attribute Types.* Web: ftp://ftp.rsa.com/pub/pkcs/.

[217] RSA Laboratories (1993). *PKCS #10:Certification Request Syntax Standard.* Web: ftp://ftp.rsa.com/pub/pkcs/.

[218] RSA Laboratories (1993). *PKCS Standard #1-#10.* Web: ftp://ftp.rsa.com/pub/pkcs/.

5.4.1 W3C Projects/Submissions

[219] P3P (1997). Platform for Privacy Preferences (formerly “P3”). Web: http://www.w3.org/.

[220] OPS (1997). Open Profiling Standard. Firefly, Netscape. Web: http://www.w3.org/.

5.5 Personal Information/Directories

[221] Versit (1995). Personal Data Interchange Specification, 1.0. Web: http://www.versit.com/.

[222] Yeong, W., T. Howes, S. Kille (1995). Lightweight Directory Access Protocol (LDAP). RFC-1777. Web: http://www.umich.edu/~rsug/ldap/doc/rfc/rfc1777.txt

5.6 Security and Payment

[223] Hickman, K. (1995). *Communications Secure Socket Layer (SSL)*. Web: http://home.mcom.com/info/SSL.html. Draft, Netscape Communications Corp.

[224] Rescorla. E., and A. Schiffman (1994). *Secure HyperText Transfer Protocol (SHTTP)*. Web: http://www.commerce.net/information/standards/drafts/shttp.txt. Draft, Enterprise Integration Technologies.

[225] Ankney, R. (1996). Enhanced Management Controls Using Digital Signatures and Attribute Certificates. ANSI X9.45, Draft 4, March 13.

[226] SET (1997). SET: Secure Electronic Transactions. VISA/MasterCard. Web: http://www.mastercard.com/set/.

[227] OTP (1997). Open Trading Protocol. Mondex. Web: http://www.mondex.com/.

[228] Intel (1997). Common Data Security Architecture (CDSA). Web: http://developer.intel.com/ial/security/cdsa/index.htm

[229] Microsoft (1997). Windows Security Support Provider Interface (SSPI). Web: http://premium.microsoft.com/isapi/devonly/prodinfo/msdnprod/msdnlib.idc?theURL=/msdn/library/sdkdoc/dpbuild_6unk.htm

[230] Sun (1997). Java Security Toolkit. Web: http://www.javasoft.com/security/whitepaper.ps

[231] Authenticode (1997). Microsoft's Authenticode system. Web: http://microsoft.com/ie/security/.

[232] Netscape (1997). Signed Applets. Web: http://www.netscape.com/.

6.0 Miscellaneous

6.1 Distributed Logic Programming

[233] Wolfson, O., and A. Silberschatz (1988). Distributed Processing of Logic Programs. *ACM SIGMOD International Conference on Management of Data* 1**7**(3), pp. 329-36.

[234] Saraswat, V., K. Kahn, and J. Levy (1990). Janus: A Step towards Distributed Constraint Programming. *Proceedings of the 1990 North American Conference on Logic Programming.* Austin, TX. pp. 431-46. MIT Press.

[235] Smith, R.G. (1980). The Contract Net Protocol: High-level Communication and Control in a Distributed Problem Solver. IEEE Transactions on Computers. Vol. C-29, 12.

6.2 Logic and Law

[236] Reiter, R. (1980). A Logic for Default Reasoning. *Artificial Intelligence* **13**(1-2), pp. 81-132.

[237] McCarty, L. Thorne (1994). Modalities over Actions, I. Model Theory. In *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning* (KR'94), pp. 437-48.

6.3 Speech Act Theory, Language/Action

[238] Searle, John (1969). *Speech Acts.* Cambridge University Press.

[239] Austin, J.L. (1962). How to Do Things With Words. Harvard University Press.

[240] Winograd, T., and F. Flores (1996). *Understanding Computers and Cognition: A New Foundation for Design.* Addison-Wesley.

6.4 Stanford InfoBus, Prototype Development

[241] Object Management Group (1995). *Object Property Service.* IBM, SunSoft, Taligent. OMG TC Document 96.6.1.

[242] Object Management Group (1993). *The Common Object Request Broker: Architecture and specification.* Accessible at ftp://omg.org/pub/CORBA/.

[243] Mac (1993). Macintosh Human Interface Guidelines. Apple Computer, Cupertino.

[244] Paepcke, A. (1996). The Stanford Digital Library Interoperability Protocol (DLIOP). Technical Report, Stanford Digital Library Project, Department of Computer Science, Stanford University.

[245] Hassan, S. (1996). JYLU—an ILU run-time kernel in Java. Web: http://db.stanford.edu/~hassan/.

[246] Janssen, B. *et al.* (1997). ILU: Inter-Language Unification. Web: http://parc.xerox.com/.

[247] Baldonado, M., K. Chang, L. Gravano, and A. Paepcke (1996). The Stanford Digital Library Metadata Architecture. Technical Report, Stanford Digital Library Project, Department of Computer Science, Stanford University.

[248] Paepcke, A., S. Cousins, H. Garcia-Molina, S. Ketchpel, M. Roscheisen, and T. Winograd (1996). Towards Interoperability in Digital Libraries: Overview and Selected Highlights of the Stanford Digital Library Project. *IEEE Computer,* 29 (5), May 1996, 61-68.

[249] Cousins, S. (1997). DLITE: The Digital Library Integrated Task Environment. Web: http://dlite.stanford.edu/.

[250] Cousins, S, S.. Hassan, A. Paepcke, and T. Winograd (1996). A Distributed Interface to the Digital Library. Technical Report, Digital Library Project, Stanford University.

[251] Ketchpel, S., *et al.* (1996). U-PAI: The Stanford Universal Payment Application Interface. Economics Subgroup, Stanford Digital Libraries Project. In *USENIX 96—-Electronic Commerce.*

[252] Baldonado, M., K. Chang, L. Gravano, and A. Paepcke (1996). The Stanford Digital Library Metadata Architecture. Technical Report, Digital Library Project, Stanford University.

[253] Röscheisen, M. *et al.* (1997). The Stanford InfoBus and Its Service Layers: Augmenting the Internet by Higher-Level Information Management Protocols. Web: http://diglib.stanford.edu/rmr/.

[254] Paepcke, A., S. Cousins, H. Garcia-Molina, S. Ketchpel, M. Röscheisen, and T. Winograd (1996). Towards Interoperability in Digital Libraries. *IEEE Computer,* 29 (5).

[255] Baldonado, M. (1997). SenseMaker: An Information-Exploration Interface Supporting the Contextual Evolution of a User's Interest. *Computer-Human Interaction Conference CHI'97*, Atlanta.

[256] Baldonado, M., K. Chang, L. Gravano, and A. Paepcke (1997). The Stanford Digital Library Metadata Architecture. *International Journal of Digital Libraries*, **1**(2).

[257] Baldonado, M., K. Chang, L. Gravano, and A. Paepcke (1997). Metadata for Digital Libraries: Architecture and Design Rationale. *Proceedings of DL'97*.

[258] Chang, C.-C., K., H. Garcia-Molina, and Andreas Paepcke (1996). Boolean Query Mapping Across Heterogeneous Information Sources. *IEEE Transactions on Knowledge and Data Engineering*, **8**(4):515-521, August.

[259] Cousins, S., A. Paepcke, T. Winograd, E.A. Bier, and K. Pier (1996). The Digital Library Integrated Task Environment (DLITE). *Proceedings of DL'97*.

[260] Gravano, L., K. Chen-Chuan Chang, H. Garcia-Molina, and A. Paepcke (1996). STARTS: Stanford Protocol Proposal for Internet Retrieval and Search. Accessible at http://www-db.stanford.edu/~gravano/starts.html

[261] Gravano, L., H. Garcia-Molina, and A. Tomasic (1994). The effectiveness of GLOSS for the text-database discovery problem. *Proceedings of SIGMOD'94*.

[262] Gravano, L., K. Chen-Chuan Chang, Hector Garcia-Molina, and Andreas Paepcke (1996). STARTS: Stanford Proposal for Internet Meta-Searching. *Proceedings of SIGMOD'97*.

[263] Paepcke, A. (1996). InterBib. Cf. http://www-db.stanford.edu/~testbed/.

[264] Balabanovic, M. and Y. Shoham (1997). Combining Content-Based and Collaborative Recommendation. *Communications of the ACM*, 40(3), March.

[265] Shivakumar, N., and Hector Garcia-Molina (1995). SCAM: A Copy Detection Mechanism for Digital Documents. *Proceedings of DL'95.*

[266] USMARC (1994). Format for Bibliographic Data: Including Guidelines for Content Designation. Cataloging Distribution Service, Library of Congress, Washington, D.C.

[267] GILS (1996). Government Information Locator Service. Accessible at http://info.er.usgs.gov:80/gils/.

[268] Hardy D.R., M.F. Schwartz, and D. Wessels (1996). Harvest User's Manual. Accessible at http://harvest.transarc.com/-afs/-transarc.com/-public/-trg/-Harvest/-user-manual/.

[269] Lagoze, C., and D. Ely (1995). Implementation Issues in an Open Architectural Framework for Digital Object Services. TR95-1590, Cornell University.

[270] Lagoze, C., and C.A. Lynch and Ron Daniel Jr. (1996). The Warwick Framework: A Container Architecture for Aggregating Sets of Metadata. TR96-1593, Cornell University.

[271] Z3950 (1995). Information Retrieval: Application Service Definition and Protocol Specification. ANSI/NISO. April.

[272] Borenstein, N., and N. Freed (1993). MIME: Multipurpose Internet Mail Extensions: Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies. Internet RFC 1521.

6.5 Selected Prior Publications

[273] Röscheisen, M. and T. Winograd (1996). A Network-Centric Design for Relationship-based Security and Access Control. Overview of the Security Architecture of the Stanford Integrated Digital Libraries Project. Invited Contribution to the *Journal of Computer Security.* Web: http://diglib.stanford.edu/rmr/.

[274] Röscheisen, M., and T. Winograd (1997). The FIRM Framework for Interoperable Rights Management: Defining a Rights Management Service Layer for the Internet. *Forum on Technology-based Intellectual Property Management*, Washington, DC. Interactive Media Association, White House Economic Council, and White House Office of Science and Technology.

[275] Röscheisen, M. and T. Winograd (1996). A Communication Agreement Framework for Access/Action Control. In *Proceedings of the IEEE Symposium on Research in Security andPrivacy*, Oakland.

[276] Röscheisen, M. (1996). Beyond Privacy as Anonymity: Rights Management Technologies for Privacy and Intellectual Property Control. Lunchtime presentation at *Computers, Freedom, and Privacy*, CFP96, Boston, March 28-30. Slides: http://pcd.stanford.edu/rmr/CFP96

[277] Röscheisen, M. (1995). General Certificates. Working Paper #12. Stanford Integrated Digital Libraries Project, Stanford University.

[278] Röscheisen, M., C. Mogensen, and T. Winograd (1994). Shared Web Annotations As A Platform for Third-Party Value-Added Information Providers: Architecture, Protocols, and Usage Examples. Technical Report, Stanford Integrated Digital Library Project, Computer Science Department, Stanford University, November 1994. Also: P*roceedings of the Third International World-Wide Web Conference*, Darmstadt, Germany; *Proceedings of CHI95*, Denver, CO. Web: http://pcd.stanford.edu/COMMENTOR/.

[279] Kamiya, K., M. Röscheisen, and T. Winograd (1995). Grassroots: Providing a Uniform Framework for Communicating, Sharing Information, and Organizing People. Short paper, *CHI '95 Conference*; Paper, *6th WWW Conference*, Paris; and Technical Report, Computer Science Department, Stanford University. Web: http://pcd.stanford.edu/Grassroots/.