Semantic Email: Adding Lightweight Data Manipulation Capabilities to the Email Habitat

Oren Etzioni, Alon Halevy, Henry Levy, and Luke McDowell
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195 U.S.A
{etzioni,alon,levy,lucasm}@cs.washington.edu

ABSTRACT

The Semantic Web envisions a portion of the World Wide Web in which the underlying data is machine understandable and applications can exploit this data for improved querying, aggregation, and interaction. This paper investigates whether the same vision can be carried over to the realm of email, the adjacent information space in which we spend significant amounts of time.

We introduce a general notion of semantic email, in which email messages consist of a database query or update coupled with corresponding explanatory text. Semantic email opens the door to a wide range of automated, email-mediated applications. In particular, this paper introduces a class of *semantic email processes*. For example, consider the process of sending an email to a program committee, asking who will attend the PC dinner, automatically collecting the responses, and tallying them up. We describe a formal model where an email process is modeled as a set of updates to a data set, on which we specify certain constraints. We then describe a set of inference problems that arise in this context, and provide initial results on some of them. In particular, we show that it is possible to automatically infer which email responses are acceptable w.r.t. a set of ultimately desired constraints. Finally, we describe a first implementation of semantic email, and outline several research challenges in this realm.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Miscellaneous;
F.2 [**Analysis of Algorithms and Problem Complexity**]: Miscellaneous;
D.4.3 [**Communications Applications**]: Electronic mail

Keywords

Semantic web, email, formal model, inference, ultimate satisfiability, lightweight data manipulation

International Workshop on the Web and Databases (WebDB).
June 12–13, 2003, San Diego, California.

1. INTRODUCTION

There is currently significant interest in making portions of the WWW machine understandable as part of the broad vision known as the "Semantic Web" [1]. While the WWW is a rich information space in which we spend significant amounts of time, many of us spend even more time on email. In contrast to the WWW, where most of our interactions involve consuming data, with email we are both creating and consuming data. With the exception of the generic header fields associated with each email message, the email information space has no semantic features whatsoever. While the majority of email will remain this way, this paper argues that adding semantic features to email offers tremendous opportunities for payoff in productivity while performing some very common tasks. To illustrate the promise, consider several examples.

- In the simplest case, suppose you send an email with a talk announcement. With appropriate semantics attached to the email, sending the announcement can also result in automatically (1) posting the announcement to a talks web site, and (2) sending a reminder the day before the talk.
- Suppose you are organizing a PC meeting, and you want to know which PC members will stay for dinner after the meeting. Currently, you need to send out the question, and compile the answers *manually*, leafing through emails one by one. Furthermore, you need to do so every few days in order to find out who has answered and who has not. With semantic email, the PC members can provide the answer in a way that can be interpreted by a program and compiled properly. In addition, after a few days, certain PC members can be reminded to answer, and those who have said they're not coming to the PC meeting need not be bothered with this query at all.
- As a variant of the above example, suppose you are organizing a *balanced potluck*, where people should bring either an appetizer, entree or dessert, and you want to ensure that the meal is balanced. In addition to the features of the previous example, here semantic email can help ensure that the potluck is indeed balanced by examining the answers and requesting changes where necessary.
- As a final example, suppose you want to give away tickets to a concert that you cannot use. You would like to send out an announcement, and have the semantic email system give out the tickets to the first respon-

dents. When the tickets are gone, the system should respond politely to subsequent requests. Alternatively, you may want to sell the tickets to the highest bidder and have the system help you with that task.

These examples are of course illustrative rather than exhaustive. However, the examples suggest a general point: we often use email for tasks that are reminiscent of lightweight data collection, manipulation, and analysis tasks. Because email is not set up to handle these tasks effectively, accomplishing them manually can be tedious, time-consuming, and error-prone.

In general, there are at least three ways in which semantics can be used to streamline aspects of our email habitat:

1. **Update:** we can use an email message to add data to some source (e.g., a web page, as in our first example).
2. **Query:** email messages can be used to *query* other users for information. Semantics associated with such queries can then be used to automatically answer common questions (e.g., asking for my phone number or directions to my office).
3. **Process:** we can use semantic email to manage simple but time-consuming processes that we currently handle manually.

The techniques needed to support the first two uses of semantic email depend on whether the message is written in text by the user or formally generated by a program on the sender's end. In the user-generated case, we would need sophisticated methods for extracting the precise update or query from the text. In both cases, we require some methods to ensure that the sender and receiver share terminologies in a consistent fashion.

This paper focuses on the third use of semantic email to streamline processes, as we believe it has the greatest promise for increasing productivity and is where the most pain is currently being felt by users. Some hardcoded email processes, such as the meeting request feature in Outlook, invitation management via *Evite*, and contact management via *GoodContacts*, have made it into popular use already. Each of these commercial applications is limited in its scope, but validates our claim about user pain. Our goal in this paper is to sketch a *general* infrastructure for semantic email processes. Feature rich email systems such as Microsoft's Outlook/Exchange offer forms and scripting capabilities that could be used to implement some email processes. However, it is much harder for casual users to create processes using arbitrary scripts, and furthermore, the results would not have the formal properties that our model provides. We describe these properties in Section 3.

To the best of our knowledge, this paper is the first to articulate and implement a general model of *semantic email processes* (SEPs).[1] Our technical contributions are the following. We first describe a formal model for semantic email processes. The formal model specifies the meaning of semantic email processes and exposes the key implementation challenges. We then pose several fundamental inference problems that can be used to boost semantic email creation and processing, and describe some initial results on these problems. We discuss implementation issues that arise for semantic email and how we have addressed these in our first semantic email prototype. Finally, we outline several research challenges related to semantic email.

[1]Associating semantic content with mail has been proposed before [7, 3], but such proposals have focused only on using semantics to improve mail search, sorting, and filtering.

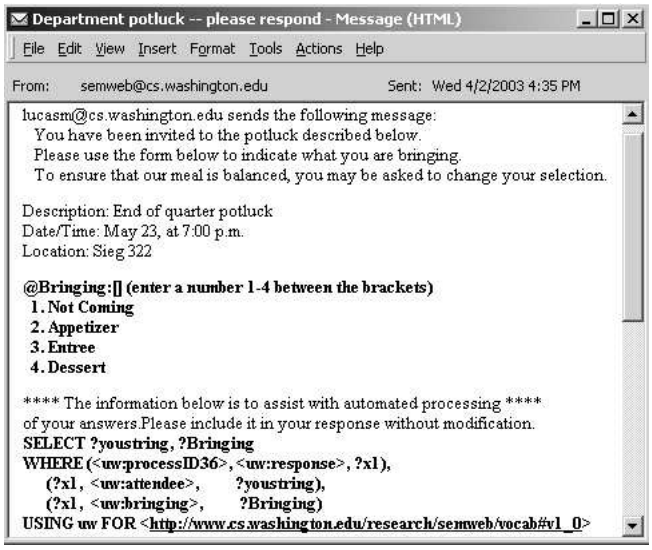

Figure 1: A message sent to recipients in a "Balanced potluck" process. The bold text at the top is a form used for human recipients to respond, while the bold text at the bottom is a query that maps their textual response to a formal language.

2. FORMAL MODEL OF SEMANTIC EMAIL PROCESSES

Our formal model of SEPs serves several goals. First, the model captures the exact meaning of semantic email and the processes that it defines. Second, the model clarifies the limitations of SEPs, thereby providing the basis for the study of different variations with varying expressive powers. Finally, given the model, we can pose several formal inference problems that can help guide the creation of semantic email processes as well as manage their life cycle. We emphasize that the users of SEPs are not expected to understand the formal model or write specifications using it.

In our model, we assume that email addresses uniquely determine individuals or sets of potential participants in the process. A SEP is initiated by an individual participant, called the *originator*. Informally speaking, the coordination of a process is achieved by a *supporting data set*, and a set of updates that the recipients of the email can make to this data set as they respond to the original email.

EXAMPLE 2.1. *We first illustrate our formal model with the example of setting up a balanced potluck. The originator of the process will initially send out a message announcing the potluck and asking everyone to say whether they are coming and whether they are bringing an appetizer, entree, or dessert (see Figure 1). As a result of this email, a supporting data set is automatically created. The data set includes a single table with a row for every recipient of the email. The table will have two columns,* email *and* bringing. *The first column will be an email address of one of the recipients, and the second column will have a constraint specifying that it can only have one of four values:* { `not-coming, appetizer, entree, dessert` }. *Initially, the second column will be set to NULL.*

The process will specify that the result of a response from a recipient p is an update to the data set, where p is allowed to update only the row where the first column is p, and the update can only modify the second column. Hence, the only thing p can do is modify NULL to one of the four allowable values.

To guarantee that the potluck is balanced, the originator will also impose a constraint on the contents of the data set. For example, the originator may specify that the difference in the number of appetizers, entrees, or desserts can be at most two.

There are still several choices that the originator needs to make. The first is whether the constraints are imposed as the responses come in (e.g., after two desserts have arrived, the system will not accept another dessert until at least one entree and one appetizer have been volunteered). Alternatively, the constraints may be imposed at the end with another series of balancing emails (and here we need to assume some cooperation on behalf of the participants). Another choice is whether to allow participants to modify their selection, and how to handle such modifications. Finally, the originator needs to decide whether and which followup messages will be sent during the process. Section 3 elaborates on some of these choices and demonstrates how inference can be used to automatically direct constraint application and message generation. □

We now define the components of the formal model: participants, supporting data set, messages, responses, and the constraint language.

Participants: a process has an originator, p_0, and a set of recipients, $\mathcal{P}$. Note that p_0 could be in $\mathcal{P}$.

Supporting data set: we assume that the supporting data set, D, is a set of relations. The initial contents of the relations are specified by the originator in the beginning of the process (usually to be a set of default values for the columns). With each relation in D we associate a schema that includes:

- A relation name and names, data types, and range constraints for the attributes. A special data type is emailAddress, whose values are the set $\mathcal{P}$. Attributes may also be given default values.
- For every relation in D, there may be a single attribute of type emailAddress that is distinguished as a from attribute, which means that rows in the relation whose value is p can only result from messages from the participant p. The from attribute may be declared unique, in which case every recipient can only affect a single row in the table.
- A set of constraints on D, denoted by C_D, specified in the constraint language we describe shortly.

Messages: Processes proceed via a set of messages M. M includes the originator's first email, asking the queries for the process. Other messages in M can either be to the recipients (or subset thereof) or to the originator. A message to the recipients is specified by:

- a time point or a triggering condition,
- a set of recipients specified by a query on D, and
- a message text, e.g., a prompt for the questions being asked by the process or a reminder to a recipient to respond to an earlier request.

A message to the originator has the first and third components above.

Responses: the set of responses to the originator's email is specified as follows:

- Attributes: the set of attributes in D that are affected by responses from recipients. This set of attributes *cannot* include any from attributes.
- Insert or Update: recipients can only add tuples, only modify tuples, or both. Recall that if there is a from field then all changes from p pertain only to a particular set of tuples.
- Single or Many: can recipients send a single response or more than one? As we explain in Section 3.1, some responses may be *rejected* by the system. By Single, we mean one non-rejected message.

Constraint language: The constraints C_D are specified in a language that includes conjunction and disjunction of atomic predicates. Atomic predicates compare two terms, or a term with a set. Terms have the following form:

- an attribute variable (referring to the value of a particular attribute in a row)
- a constant
- an aggregate applied to a column of a relation, or to a subset of the rows that satisfy an equality predicate (thereby obtaining the aggregate value of a single group in the relation).

We allow comparison predicates ($=, \neq, <, \leq$), LIKE, and $\in$, $\notin$ between a constant and an enumerated finite set.

EXAMPLE 2.2. *As described earlier, in the balanced potluck example, we will have a single table named* Potluck *with two columns:* email, *of type* `emailAddress` *and declared to be unique, and* bringing, *with the constraint* Potluck.bringing $\in$ { `not-coming, appetizer, entree, dessert` }. *The default value for the* bringing *attribute is NULL.*

In addition to the single column constraint, we will also specify several constraint formulas similar to the one below, specifying that the potluck should be balanced (please note that the below is not a proposal for a syntax of the constraint language so, for now, it should be taken with kindness):

*(Count * where* bringing = 'dessert') $\leq$
*(Count * where* bringing = 'appetizer') + 2

Let us assume for now that except for the range constraint on the value of the bringing *attribute, there are no additional constraints on responses. Furthermore, assume that recipients are allowed to send a single (acceptable) response, and hence cannot change their mind after they've committed to a dish.*

Finally, the set of messages in our example includes (1) the initial message announcing the potluck and asking what each person is bringing, (2) messages informing each responder whether their response was accepted or not, (3) a reminder to those who have not responded 2 days before the potluck, (4) regular messages to the originator reporting the status of the RSVPs, and (5) a message to the originator in the event that everyone has responded. □

3. INFERENCE FOR SEMANTIC EMAIL

Given the formal model for a SEP we can now pose a wide variety of inference problems, whose results can serve to assist in the creation and management of SEPs. The data-centric flavor of our model will enable us to bring various techniques from data management to bear on these inference problems.

The core problem we want to address using inference is whether a SEP will terminate in a *legal* state, i.e., a state that satisfies C_D. The input to the inference problem includes the constraints C_D and possibly the *current* state of D along with a response r from a recipient. The output of the inference problem is a condition that we will check on r to determine whether to *accept* r. In our discussion, we assume that r is a legal response, i.e., the values it inserts into D satisfy the range constraints on the columns of D.

The space of possible inference problems is defined by several dimensions:

- **Necessity vs. possibility:** as in modal logics for reasoning about future states of a system [14, 8, 4], one can either look for conditions that guarantee that *any* sequence of responses ends in a desired state (the $\Box$ operator), or that it is possible that some sequence of responses brings us to a desired state (the $\Diamond$ operator).
- **Assumptions on the recipients:** in addition to assuming that all responses are legal, we can consider other assumptions, such as: (1) *all* the recipients will respond to the message or (2) the recipients are flexible, i.e., if asked to change their response, they will cooperate.
- **The type of output condition:** at one extreme, we may want a constraint C_r that can be checked on D when a response r arrives, where C_r is specified in the same language used to specify C_D. At another extreme, we may apply an arbitrary procedure to D and r to determine whether r should be accepted. We note that a constraint C_r will inevitably be weaker than an arbitrary algorithm, because it can only inspect the state of D in very particular ways. As intermediate points we may consider constraints C_r in more expressive constraint languages. Note that in cases where we can successfully derive C_r, we can use database triggers to implement modifications to D or to indicate that r should be rejected.
- **Generation of helpful intermediate messages:** in addition to finding a condition for accepting responses, we may infer helpful messages when a response is rejected (e.g., to suggest that while a dessert could not be accepted an entree would be welcome).

As a very simple example, suppose we consider the case where we want *all* response sequences to end in a legal state, we make no assumptions on the recipients, and we are interested in deriving a constraint C_r that will be checked when a response arrives. If the initial state of D is a legal state, then simply setting C_r to be C_D provides a sufficient condition; we only let the data set D be in states that satisfy C_D. In the example of the balanced potluck, we will not accept a response with a dessert if that would lead to having 3 more desserts than entrees or appetizers.

In some cases, such a conservative strategy will be too restrictive. For example, we may want to continue accepting desserts so long as it is still *possible* to achieve a balanced potluck. This leads us to the following inference problem.

3.1 Ultimate Satisfiability

We now describe our central result concerning inference for SEPs. Our goal is to find the *weakest* condition for accepting a response from a recipient. To do that, we cut across the above dimensions as follows. Suppose we are given the data set D after 0 or more responses have been accepted, and a new response r. Note that D does not necessarily satisfy C_D, either before or after accepting r. We will accept r if it is *possible* that it will lead to a state satisfying C_D (i.e., considering the $\Diamond$ temporal operator). We do not require that the condition on r be expressed in our constraint language, but we are concerned about whether it can be efficiently verified on D and r. Furthermore, we assume that recipients can only update their (single) row, and only do so once. Hence, the columns that can be affected by the recipients start out with the NULL value, and can get assigned values in $a_1, \ldots, a_n$.[2]

DEFINITION 3.1. *(**ultimate satisfiability**) given a data set D, a set of constraints C_D on D, and a response r, we say that D is ultimately satisfiable w.r.t. r if there exists a sequence of responses from the recipients, beginning with r, that will put D in a state that satisfies C_D.* $\Box$

In what follows, let $\mathcal{C}^\wedge$ be our constraint language restricted to conjunctions of atomic predicates (i.e., disjunction is not allowed, and negation can only be applied on atomic predicates). A term in a predicate of C_D may select a group of rows in an attribute A, and aggregate the value of the corresponding values in an attribute B. We say that the aggregation predicates in C_D are *separable* if whenever there is a non-COUNT aggregate over an attribute B, then B is not a grouping attribute in any term. (All of the examples given in this paper can be expressed with separable constraints.) We consider the aggregation functions MIN, MAX, COUNT, SUM, and AVERAGE.

THEOREM 3.1. *Let $\mathcal{S}$ be a semantic email process where C_D is in the language $\mathcal{C}^\wedge$. Then,*

- *If the predicates in C_D are separable, then ultimate satisfiability is in polynomial time in the size of D and C_D.*
- *If the predicates in C_D are not separable, then ultimate satisfiability is NP-hard in the size of D.* $\Box$

As an example of applying this theorem, in the balanced potluck example, suppose a new dessert response arrives. At that point, the inference procedure will (1) determine the maximal number of people who *may* come to the potluck (i.e., the number of recipients minus the number of people who replied `not-coming`), (2) check that even if the dessert response is accepted, then there are still enough people who have not answered such that the ultimate set of dishes could be balanced.

Discussion: The challenge in proving this theorem is in reasoning about the possible relationships between aggregate values (current and future), given a particular state of D. Reasoning about aggregation has received significant attention in the query optimization literature [15, 17, 9, 10, 2, 5]. This body of work considered the problem of optimizing queries with aggregation by moving predicates across query blocks, and reasoning about query containment and satisfiability for queries involving grouping and aggregation. In contrast, our result involves considering the current state of the database to determine whether it can be brought into a state that satisfies a set of constraints. Furthermore, since C_D may involve several grouping columns and aggregations, they cannot be translated into single-block SQL queries, and

[2] As it turns out, the case in which the recipients can add rows is actually easier, because there are less constraints on the system.

hence the containment algorithms will not carry over to our context.

To the best of our knowledge, formalisms for reasoning about workflow [13, 12] or about temporal properties of necessity and possibility have not considered reasoning about aggregation. For instance, workflow formalisms have generally been restricted to reasoning about temporal and causality constraints [16].[3] One exception is the recent work of Senkul et al. [16], who expand workflows to include *resource constraints* based on aggregation. Each such constraint, however, is restricted to performing a single aggregation with no grouping (and thus could not express the potluck constraint given in Example 2.2). In addition, their solution is based upon general constraint solving and thus may take exponential time in the worst case. We've shown, however, that in our domain SEPs can easily express more complex aggregation constraints while maintaining polynomial inference complexity in many interesting cases.

Ultimately the problem of reasoning about semantic email will be related to the problem of reasoning about e-services; see [6] for a recent survey.

4. IMPLEMENTATION AND USABILITY ISSUES

There are several significant challenges involved in implementing semantic email and getting it adopted widely. Our formal model provides a framework for identifying these challenges. Below we discuss some of these challenges, and describe the particular choices we made in our prototype implementation.

Message handling: The first and most important challenge is how to manage the flow of messages. The problem can be divided into three: message creation, transport, and reply. For message creation, we envision a set of GUIs that guide the user through composing the message and selecting the appropriate options for the process. In addition, there will be an interface for monitoring the progress of the process. Under the covers, these interfaces can make use of the inference procedures described above.

Ideally, these GUIs would be integrated with the user's mail client, which would then handle sending and receiving process-related mail on the user's behalf. For replying, a recipient's email client would present the recipient with an interface for constructing legal responses, or automatically respond to messages it knows how to handle (e.g. "Decline all tickets to the opera"). This approach, however, requires all participants in a process to install additional software (limiting its applicability) and is complicated by the variety of mail clients currently in use.

Consequently, we have adopted the following pragmatic strategy. Users originate a process by selecting a simple text form from a shared folder or a webpage, filling out the form, and then mailing that form to a special email address (e.g., `semweb@cs.washington.edu`). A central server reads this message, creates a new email process based on the parameters in the form, and sends out an initial message to the recipients. These messages also contain a simple text form that can be handled by any mail client. Recipients reply via mail[4] directly to the server, rather than to the originator, and the originator receives status and summary messages directly from the server when appropriate. The originator can query or alter the process via additional emails or a web interface. This approach is simple to implement, requires no software installation, and works for all email clients.[5] We believe that divorcing the processing of semantic email (in the server) from the standard email flow (in the client) will facilitate adoption by ameliorating potential user concerns about privacy and about placing potentially buggy code in their email client.

Human/Machine Interoperability: An important requirement is that every message must contain both a human-understandable (e.g. "I'm giving away 4 opera tickets") and an equivalent machine-understandable portion (e.g., in RDF or SQL). For messages sent to a recipient, this ensures that either a human or a machine may potentially handle and respond to the message. Thus, a process originator may send the same message to all recipients without any knowledge of their capabilities. For responses, a human-readable version provides a simple record of the response if needed for later review. In addition, a machine-understandable version enables the server to evaluate the message against the process constraints and take further action.

In our implementation, we meet this requirement by having the server attach RDF to each outgoing text message. A human responds by filling out an included text form, which is then converted into RDF at the server with a simple mapping from each field to an unbound variable in a RDQL query associated with the message.[6] A machine can respond to the message simply by answering the query in RDF, then applying the inverse mapping in order to correctly fill out the human-readable text form.

Steamlining semantics with other aspects of email: Despite the advantages of semantic email, we do not want to create a strict dichotomy in our email habitat. In our potluck example, suppose that one of the recipients wants to know whether there is organized transportation to the potluck (and this information affects his decision on what to bring). What should he do? Compose a separate non-semantic email to the originator and respond to the semantic one only later? A better solution would be to treat both kinds of emails uniformly, and enable the recipient to ask the question in replying to the semantic email, ultimately providing the *semantic* response later on in the thread.

Our implementation supports this behavior by supplying a "remark" field in each response form, where a recipient may include a question or comment to be forwarded to the originator. For a question, the originator can reply, enabling

[3]Workflow formalisms could potentially convert aggregation constraints to temporal constraints by explicitly enumerating all possible data combinations, but this may result in an exponential number of states.

[4]Alternatively, we could send recipients a link to a suitable web-based form to use for their response. This mechanism is fully consistent with our formal model and has some advantages (e.g., forms are not restricted to text, immediate data validation). We chose instead to use email because we feel it fits more naturally with how recipients typically handle incoming messages.

[5]This approach can still enable the automated answering of common questions, but requires an extra step for recipients to forward such messages to the server

[6]Technically, this response does not contain a pure machine-understandable portion (e.g. in raw RDF), but does contain all the information necessary (fields, RDQL, and mapping) to easily produce this portion.

the recipient to respond to the original semantic question with the included form, or to pose another question.

Database and inference engine: There are several system issues to consider, such as where does the actual database and inference engine reside. Right now, we built a separate semantic email system implemented as a centralized server supported by a relational database.

Implementation Status: We have developed a prototype semantic email system and deployed it for public use.[7] So far we have developed simple processes to perform functions like collecting RSVPs, giving tickets away, and organizing a balanced potluck; these can be customized for many other purposes (e.g. to collect N volunteers instead of give away N tickets). We are currently testing the system with real users to determine how well these processes generalize to everyday needs.

The prototype is integrated with our larger MANGROVE [11] semantic web system. This provides us with an RDF-based infrastructure for managing email data and integrating with web-based data sources and services. For instance, the MANGROVE calendar service accepts event information via email or from a web page. Future work will consider additional ways to synergistically leverage data from both the web and email worlds in this system.

5. CONCLUSIONS

We have introduced a paradigm for enriching our email habitat with the ability to perform lightweight data collection and manipulation tasks. We believe that this form of semantic email has the potential to offer significant productivity gains on email-mediated tasks that are currently performed manually in a tedious, time consuming, and error-prone manner. Moreover, semantic email opens the way to scaling similar tasks to large numbers of people in a manner that is not feasible with today's person-processed email. For example, large organizations could conduct surveys and voting via email with guarantees on the behavior of these processes. We motivated semantic email with several examples, presented a formal model that teases out the issues involved, and used this model to explore several important inference questions. Finally, we described our publicly accessible prototype implementation.

This short paper focused on a particular class of semantic email processes (SEPs) that can be described with a simple constraint language. However, our notion of semantic email is substantially broader, which suggests several research challenges:

- Our formal analysis considered a point in the space of semantic email processes, which led to our central theorem. It would be worthwhile to carefully analyze the remaining points in the space described in Section 3.
- In our analysis we adopted a limited constraint language — how does increasing its expressive power impact the tractability of inference?
- We considered a small set of illustrative examples. Future work will explore additional tasks and investigate any impediments to widespread adoption.
- SEPs are only one instantiation of semantic email, which far from exhausts its potential. New ways to leverage semantic email seem like a promising direction for future work.

[7]Accessible at www.cs.washington.edu/research/semweb/email

Finally, we see semantic email as a first step in a tighter integration of the web (semantic or not) and email, the two information spaces in which many of us spend the bulk of our online time.

6. ACKNOWLEDGMENTS

This research was partially supported by NSF ITR Grant IIS-0205635, by NSF CAREER Grant IIS-9985114 for Alon Halevy, and by a NSF Graduate Research Fellowship for Luke McDowell.

7. REFERENCES

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.

[2] S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. In *Proc. of PODS*, pages 155–166, 1999.

[3] D. Connolly. A knowledge base about internet mail. http://www.w3.org/2000/04/maillog2rdf/email.html.

[4] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. M.I.T Press, 1995.

[5] S. Grumbach and L. Tininini. On the content of materialzed aggregate views. In *Proc. of PODS*, 2000.

[6] R. Hull, M. Benedikt, V. Christophides, and J. Su. E-Services: A look behind the curtain. In *PODS*, 2003.

[7] A. Kalyanpur, B. Parsia, J. Hendler, and J. Golbeck. SMORE - semantic markup, ontology, and RDF editor. http://www.mindswap.org/papers/.

[8] R. E. Ladner. The computational complexity of provability in systems of model propositional logic. *SIAM Journal on Computing*, 6(3):467–480, 1977.

[9] A. Y. Levy and I. S. Mumick. Reasoning with aggregation constraints. In *Proc. of EDBT*, Avignon, France, March 1996.

[10] A. Y. Levy, I. S. Mumick, and Y. Sagiv. Query optimization by predicate move-around. In *Proc. of VLDB*, pages 96–107, Santiago, Chile, 1994.

[11] L. McDowell, O. Etzioni, S. D. Gribble, A. Halevy, H. Levy, W. Pentney, D. Verma, and S. Vlasseva. Evolving the semantic web with Mangrove. Technical Report UW-CSE-03-02-01, February 2003.

[12] C. Mohan. Workflow management in the internet age. www.almaden.ibm.com/u/mohan/workflow.pdf, 1999.

[13] S. Mukherjee, H. Davulcu, M. Kifer, P. Senkul, and G. Yang. Logic based approaches to workflow modeling and verification. In *Logics for Emerging Applications of Databases*, 2003.

[14] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[15] K. Ross, D. Srivastava, P. Stuckey, and S. Sudarshan. Foundations of aggregation constraints. In A. Borning, editor, *Principles and Practice of Constraint Programming*. Lecture Notes in Computer Science, 874. Springer Verlag, 1994.

[16] P. Senkul, M. Kifer, and I. H. Toroslu. A logical framework for scheduling workflows under resource allocation constraints. In *VLDB*, 2002.

[17] D. Srivastava, S. Dar, H. V. Jagadish, and A. Y. Levy. Answering SQL queries using materialized views. In *Proc. of VLDB*, Bombay, India, 1996.