Specifications and Architectures of Federated Event-Driven Systems *

K. Mani Chandy and **Michael Olson**
California Institute of Technology
Computer Science Department
1200 E. California Blvd.
MC 256-80
Pasadena, California, USA

Abstract

Federated event driven systems (FEDS) integrate event data pushed to it or pulled by it from multiple sites within and outside an enterprise. Performance requirements for these applications can be met by implementing applications on networks of multiprocessors. This paper describes a graph model for applications and a method of mapping the graph on to multiprocessor networks. Information flows among processors within an address space are implemented by passing pointers while information flows across address spaces are implemented by message passing. The model and implementation allows an application and the underlying hardware infrastructure to be changed relatively easily. The paper describes an application implemented in this way.

Introduction

Federated event-driven systems (FEDS) are integrations of services and data streams provided by multiple entities. A federated application gets data pushed to it by some data sources and pulls data from other sources. FEDS applications acquire and analyze heterogeneous types of data — such news stories, blogs, Twitter "twits", stock tickers, and medical, seismic and power-consumption sensors — acquired from heterogeneous sources that have very different performance characteristics. The heterogeneity of data types and the sources results in performance and specification problems. This paper proposes methods to overcome these problems and describes an implementation of an application based on these methods.

Specifications Specifying the varieties of events that appear in FEDS applications by a single event processing language does not seem tractable. One of the applications that we are working on detects damage to buildings after an earthquake. Data arrives from accelerometers as well as from text messages from users. Accelerometer data and textual data are analyzed in different ways. Since the types of components in these applications are varied, we specify an application as a composition of processing steps where the specification of each step uses the notation appropriate for that step. The specification of an application is a dataflow graph. A node of the graph is a processing step. We use a library of components where each processing step of an application is an instantiation of a component. The component is tailored to the specific requirements of a step by specifying the component's parameters.

Federated systems have been studied for over two decades in the context of database systems and application integration (Gawlick 2001). Federated systems for event processing have been researched by several groups (Branson et al. 2007). Graphical programming models have been studied even longer, and now appear in event processing systems provided by Tibco, Streambase and other companies. Research on semantics of data-flow and message-processing networks has been carried out for decades (Misra and Chandy 1981; Kahn 1974). We use a well-known model partly because we can use results developed over decades on the semantics and performance of the model.

Performance Characteristics FEDS applications have characteristics that impact their performance. These include the following.

1. **Resilience to Delays**: Different services may be invoked in different steps of processing data in an application. A service provided by a third-party provider may be invoked on a step of an application while another step may be executed by a method call on a local object. A service invocation across the Internet takes much more time than a local method call. Response times vary a great deal across providers. Therefore these applications must be resilient to large ranges of response times across components of an application.
2. **Variable Load**: Loads on these applications are often bursty. Furthermore processing steps for different components of a FEDS application may require widely different amounts of processing time.
3. **Agility**: These applications are modified over time by adding and deleting processing components and changing the dataflow graph. Incremental changes to the dataflow must be implemented easily. When hardware is added to the system, or a hardware component fails, system administrators should be able to map the application on to

*This work is funded in part by the Caltech Lee Center for Networking

the new hardware platform in a simple way.

We show how exploiting parallel processing helps to deal with these characteristics.

Parallel Architectures Quad processor computers are widely available. Costs of multiprocessor shared-address space machines with many (16 and higher) processors are likely to come down in a decade. Multicomputer networks have been shown to have high performance for some applications. We propose an implementation of event processing applications on multicomputer networks where each computer is a multiprocessor. The implementation separates the logical specification - the dataflow graph and component specification - from the parallel hardware structure while exploiting the multiprocessors within each node of the network and balancing load across the computers of the network.

Applications

In this section we describe applications that illustrate the rationale for the proposed specification mechanism and software architecture.

The Personal Information Broker A personal information broker (PIB) acquires and analyzes data from multiple sources, detects events, and sends event objects to appropriate devices such as cell phones, PDAs and computers. A PIB also displays data in multiple dashboards. A PIB may acquire data from blogs, news stories, social networks such as Facebook, Twitter, stock prices and electronic markets. In this paper we restrict attention to PIBs that analyze blogs.

An example of a PIB application is one that learns, in near-real time, changes in the numbers of posts that discuss relationships between different entities. Consider the following example from this election year. The Hilary Clinton campaign may want to know, as quickly as possible, whether the number of blog posts containing both Clinton and some other entity changes significantly. For example, as shown in Figure 1, a PIB application can detect that the number of blog posts containing Clinton and Governor Elliott Spitzer of New York changed significantly before the Governor resigned. Note that the application isn't looking specifically for an association between Clinton and Spitzer — the input to the application is only the entity, i.e., Clinton, of interest. Related entities are discovered by monitoring other entities that co-occur with the entity of interest and estimating the relevance of the co-occurance at each point in time that it is found in the stream. The changing numbers of posts containing the monitored entity and the related entity is detected automatically by the application. The challenge is to carry out this detection in near real-time as opposed to *post facto* data mining.

Information in the blog universe can be correlated automatically with other types of data such as electronic markets. For instance, when a component of a PIB detects anomalous time series patterns of blog posts associating a presidential candiate, say Senator Obama, with some other entity, say Rev. Jeremiah Wright, then it can determine whether Senator Obama's "stock" price on the Iowa Electronic Market

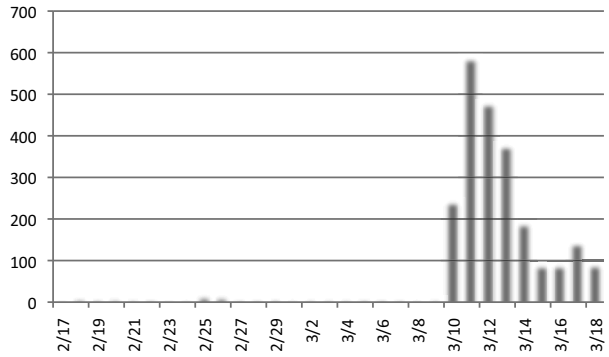

Figure 1: Volume of posts mentioning both "Clinton" and "Spitzer" over time.

changed significantly at that time. If the price of Senator Obama's stock didn't change much then we may deduce (though the deduction may be false with — we hope — low probability) that market makers don't expect this association to have much impact on the likelihood of Senator Obama's chances of winning.

Another PIB application continuously searches for entities to discover when the rate of blog posts containing names of *any* of these entities changes rapidly. This application has no input — it detects changes in the frequency of posts about any and all entities. For example, the application can discover that the volume of posts about the bank IndyMac changed significantly on July 8 and July 12 (see Figure 2). The event that is detected does not predict anything; the event is merely an anomaly in the time-series patterns of posts about any entity. In this example, the failure of IndyMac became public knowledge on July 11.

One of the challenges of developing PIB systems is the development of algorithms that do not use too much memory or computational power to analyze huge volumes of data. Consider the application that detects anomalies in time series in the numbers of posts containing the name of an entity. In just 14 weeks of data, 77 million unique entities (see *Experimental Results*) were discussed in blog posts. We do not know which entities are going to engender anomalous time-series behavior beforehand, so we have to track every single one of them. The challenge is to detect anomalous behavior without repeatedly executing queries.

Radiation Detection Nuclear radiation sources generate photons in a random (Poisson) manner. The problem is to detect pedestrians and land, air and sea vehicles carrying dangerous amounts of radiation. A sensor generates a discrete signal when it is struck by a photon. The direction of the photon is not registered on the sensor. The problem is to interdict a radiation source by using a combination of static and dynamic radiation sensors coupled with images from cameras, and intelligence data. Detection of a radiation source is based on Bayesian analysis of frequencies of photon hits and photon energies obtained by sensors and also

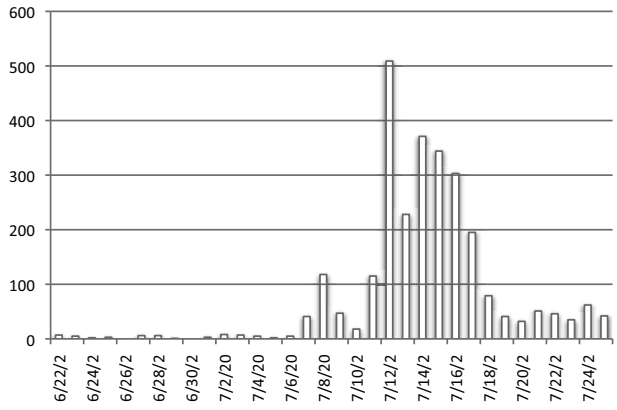

Figure 2: Volume of posts about IndyMac over time.

from other types of information (such as video images). The specification of the algorithm to detect the radiation source does not lend itself to description by standard event processing languages. Calculations are expensive and so execution on parallel architectures helps.

Electric Power Grid Management Electric utilities are deploying wireless-enabled smart meters at residences and businesses. These meters communicate periodically with the utility sending information about power consumed and applicances used during each interval. The devices also get control signals from the utility, and the devices can control appliances such as airconditioners. The problem is to correlate data from millions of smart meters to detect a probable overload situation, and then respond by temporarily turning off appliances, such as car-battery chargers, for short periods of time. This sense and respond system reduces the likelihood of blackouts.

Earthquake Shaking Intensity Map Generation Large numbers of inexpensive accelerometers connected to the Internet via computers can serve as a seismological network with thousands of sensors, albeit with a great deal of noise. Text messages generated by people after an earthquake provide valuable information for further refining damage assessments as well. The challenge is to integrate signals from accelerometers with text messages and other human-generated information that arrives after an earthquake so as to dispatch first responders to the areas where they are most needed. Information integration is based on geological models of seismic activity as well as models of human data generation after quakes. No single event processing language is adequate for specifying where and when severe shaking has occurred. Computing the shake-map is expensive, and networks of multiprocessors are used for this purpose.

Specification Notation

The specification notation we use is modeled as a directed acyclic graph where vertices represent processing steps and directed edges represent the flow of data. (Recursion can be handled by dataflow graphs, but here we restrict attention to feed-forward dataflow.) The acquisition of data is a processing step and is therefore represented by a vertex. Data may be acquired either from a data stream pushed to the application or from a service that replies to requests for data. Some processing steps may invoke services from organizations within the enterprise, some steps may invoke services outside the enterprise, and some may be computational steps within the application itself. The action of pushing data to a dashboard on a device (cellphone or computer) is also a processing step represented by a vertex.

The graph is a meta-specification in the sense that it shows the (partial) order of processing steps, but it does not specify each step. We use different notations for each step depending on the type of processing associated with the step. We use Web services notation to describe a step that invokes a Web service, and we use natural language processing specifications for steps that deal with identifying concepts and entities in English and other languages.

The specification notation we use has been used for decades in representing process flows particularly in business process management. Our analysis of the example applications presented in this paper suggests that no single event processing language is adequate for specifying all event processing applications. Therefore, we advocate a compositional multilingual approach for EPL.

Next, we evaluate the relative suitability of the compositional approach to EPL by considering each of the example applications.

Specifying the Personal Information Broker **Data Acquisition**: Data can be acquired from multiple sources – currently we use Spinn3r, later we will also acquire IEM, Twitter, Technorati, etc. Each of these acquisitions is specified differently. Acquisition of Spinn3r data, referenced in Figure 3 step 1, is achieved through changing URL arguments in a manner defined by Spinn3r. Thus, the specification is unique to Spinn3r. While that particular specification cannot be reused, using the compositional approach, exchanging Spinn3r for Twitter, a news feed, or an instant messaging account while maintaining the integrity of the composition is trivial. The specifications for all of these information interfaces are very different; a notation that allows the description of composite applications must account for this.

Processing steps: Some steps invoke services, for instance Calais. The specification for Calais follows the WSDL notation, which means that implementing a client for Calais is relatively trivial. Clients can be created using the WSDL file so that information exchanges with Calais are nearly transparent to the application. For these types of external services invoking simple functions, the WSDL notation is particularly appropriate. The specification of the service from a compositional standpoint revolves around the inputs and outputs of the service. In this case, Calais acts on plain text data and returns an RDF document with semantic annotations of that data.

Many processing steps do not invoke external services; in

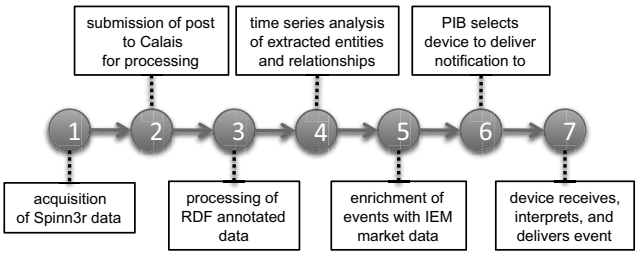

Figure 3: Example information flow using compositional approach.

Figure 3, steps 3 and 4 are internal processing steps. Step 3, which involves processing the RDF results of Calais and other semantic services, is necessary to extract the encoded information that we are interested in. Specification of this type of processing step needs to identify what classes of information are of interest. Focusing on subsets of the potential data set can greatly simplify processing considerations. Step 4, which involves time series analysis on the extracted data, uses familiar specification notations like EPL to describe what kinds of patterns in the subset of data returned by step 3 are interesting. Other operations, such as compression of the frequency stream using exponential smoothing, can also be performed, and there should be a way of specifying these operations as well.

Response: Where a PIB is used, event response is broken into two steps; those steps are represented in Figure 3 steps 6 and 7. The first stage, step 6, evaluates the event using an estimate of the current context of the intended recipient and information gleaned from past usage of the system. Based on that evaluation, the event is routed to one of several possible response devices, discarded, or held until further information can be acquired. Specification of how context impacts event delivery needs its own set of rules.

The second stage, step 7, is the receipt by the device. The resultant event must be encoded in such a way that the device can answer questions it might encounter, such as: which dashboard should the event be displayed on; should notification of event arrival be audible or visual; and if the user wants more information, where should they look. For this, a notation that thoroughly describes events such that a recipient application can accurately interpret how to handle the incoming events is necessary.

Given the wide variety of domains covered by a PIB application, no individual notation can accurately describe how each component should perform its task. Instead, where notations are appropriate to a particular domain, they should be used for expressing sub-tasks in that domain.

Specifying a Radiation Detection System An intelligent radiation sensor system (IRSS) consists of networks of static and mobile intelligent personal radiation locators (IPRLs). An IPRL has a crystal that generates an electrical signal when struck by a photon with energy within a specified range. An IPRL also has sensors that identify its location and its orientation in 3D. The output of an IPRL is a sequence of events where an event is the generation of an electrical signal, and a continuous stream of data identify location and orientation. IPRLs communicate with each other and with base stations through wireless. Mobile IPRLs collaborate by sending their information to each other subject to bandwidth constraints.

Each IPRL has an *a priori* distribution for the number (possibly zero) and locations of radiation sources in a region of interest such as a field where a political rally is being held or a ship that is being searched for nuclear material by a boarding party. The specification of the system includes specifications of the updates to the distributions calculated by each IPRL and base stations based on information sensed by the IPRL itself and information communicated by others. Events that are aggregated by the system may include camera data and textual information. The meta-specification of information flow for each IPRL and the base station can be represented by a dataflow graph. The specification of each processing step is given by a collection of equations for updating distributions and for movement of mobile detectors. These equations cannot be represented conveniently in the EPLs that have been proposed in the literature. A multilingual approach allows for the appropriate methods to be used for different components of the system.

Specifying an Electricity Grid Management System This example limits attention to only a tiny part of the overall grid management system: control of appliances in houses. The appliances of interest in Southern California and other warm regions are primarily airconditioners, chargers for hybrid and electric cars, and electric clothes dryers. Event processing nodes receive streams of sensor data from smart meters in households served by common infrastructure elements such as transformers. The nodes analyze sophisticated models of the grid to determine whether appliances need to be turned off or turned back on. The system also accesses databases to determine contract policies of customers to identify which households permit appliances to be turned off by the utility and for how long. For this application the flow of information in steps is represented by an acyclic graph, but the notations for specifying each node vary — some node specifications use SQL and others use programs that model the grid.

Software and Hardware Architectures

The Computational Graph The software structure appropriate for these applications is essentially the graph of dataflow with optimizations for efficiency. We call the processing elements of the applications *nodes* rather than vertices to distinguish the actual computational operations from the logical computational steps in the dataflow.

A node of the computational graph is shown in Figure 4; it receives streams of data on one or more input queues. A data item either describes a single event, such as a single blog post, or a group of events such as a collection of blog posts; performance considerations determine the data granularity. There is insufficient space for a detailed discussion of performance issues; the point to be made is that data items

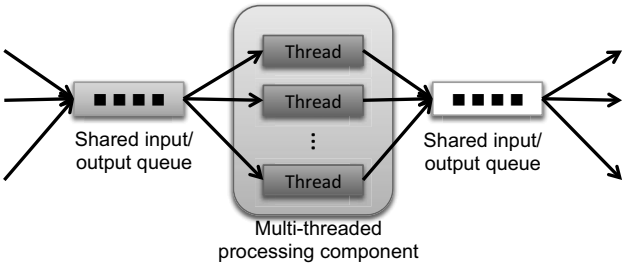

Figure 4: Example structure of a *node*, or processing component.

may arrive in aggregated units or be aggregated before they are sent from node to node.

A generic node outputs data items into input queues of other nodes. The interconnection of nodes corresponds to that of vertices in the dataflow graph. Each node is an isolated module with input queues and output ports. Nodes are connected by connecting an output port of one node to an input queue of another.

Data Items as Objects or Serialized Objects A data item in an input queue of a node is an object (or more accurately a pointer to memory where data about the object is stored). The data that flows along an edge is either an object or is a message containing a serialized copy of an object. Data items that flow between nodes within the same process are objects; by contrast, data items that flow across processes are serialized objects. Passing an object from a node to a queue within the same process only requires putting a pointer into the queue, which is a fast operation. Serializing and deserializing objects are computationally much more expensive actions compared to the action of passing an object to a queue

The computation graph is independent of how nodes are mapped to processes. This mapping is determined at a later stage in the design to obtain desired performance. When an edge of the computation graph crosses process boundaries, serializaion and deserialization take place, but the nodes themselves remain unchanged.

Thread Pools A generic node has a pool of threads. If an input queue is not empty and an idle thread is available in the pool then the thread takes the next data item in an input queue. The execution of the thread may be specified in UML or some graph abstraction or in a programming language. One of the steps to be executed at a node may be an invocation of a service. Many threads may access a service concurrently. For example, many threads access the Calais service at the same time. Services are usually implemented with thread pools; thus multiple invocations of the same service are processed concurrently. For example, consider a case where the response time (latency) for a service is 1 second, and data items arrive in input queues at the average rate of 10 per second. Since the service is itself implemented using a thread pool it can service 10 requests per second giving each request an average response time of 1 second. Therefore, a thread pool of size greater than 10 (to deal with burstiness) at the node helps manage the large and variable latency of services that are invoked. In our implementations we use thread pools with about 20 threads to overcome variability in response times (see *Experimental Results*). We plan to investigate the optimum sizes of thread pools through experiment and mathematical models.

Messaging Middleware An Enterprise Service Bus (ESB) can be used as the infrastructure for communicating serialized objects across processes. An analogous bus is used for communicating (references to) objects between nodes within the same process. The mapping of computational nodes to processes determines whether an intra-process local bus is used to communicate objects or whether an inter-process ESB is used.

Parallel Hardware The graph model for the meta specification, coupled with the software architecture represented by object flows between computational nodes, is ideal for implementation on clusters of multicore processors. Thread pools in computational nodes exploit multiple cores. When processor utilization gets high, a new processor is acquired, processes are instantiated on the new processor, and some computational nodes are moved from their current processes to the new processes. The specification model and the computational model remain unchanged; all that changes is the mapping of computational nodes to processes. In our current implementation, this remapping requires the system to be recompiled. Later we plan to explore constructs, such as reflection, to allow incremental modification of the system without recompiling and restarting it.

Experimental Results

The first three steps of the application described by the information flow in Figure 3 were implemented; a database is used to store processed data for later processing to analyze missed event opportunities and run simulations of input streams. Using this data, algorithms for detecting events in the stream are being developed to help complete the implementation of step 4.

The stream itself is relatively large; over 14 weeks of data, the database has increased to 164GB, or about 1.7GB per day. The average number of posts for a given weekday is 292,879, but, being bursty in nature, the number of posts in a timeframe varies dramatically, from 0 messages in a given second to upwards of 30. Weekends see only 76% of the traffic that weekdays see, or an average of 223,124 posts. Algorithms that smooth frequencies over time need to account for the discrepancies in post volumes between weekdays and weekends or holidays; without accounting for this discrepancy, erroneous events can be generated by an algorithm which incorporates that discrepancy as part of its algorithm for detecting significant changes.

It is worth noting that, out of a sample of 31,304,036 processed posts, only 21,741,604 posts actually entered the database. The singular criterion for entering the database is that Calais has identified at least one entity that the post

referred to. This indicates that only roughly 70% of the processed posts contain any meaningful content that can be analyzed for generating events. The ability to identify and discard the 30% of posts that are meaningless before they are processed would yield large performance gains, both locally for the corresponding process and remotely for the Calais web service.

Calais, by Thomson Reuters, is an excellent example of the kinds of latencies and error rates that compositional applications can expect when integrating web services into their information flow. After processing 31,304,036 posts, latency measured from beginning submission to completion of the receipt of the result averaged only 733 milliseconds. Fluctuations were high, ranging from only marginally more than round trip estimates-in the 100 ms range-to a minute to process more complex posts. With proper threading, latency had no impact on system throughput, which at peak utilization reached 30 posts per second, our maximum allowable throughput by Calais. Error rates were, for our purposes, very reasonable; overall error averaged only 0.66% of the total volume of posts, which includes errors from posts that are too large to process and non-English posts that slipped through Spinn3r's language filter.

One major difficulty in tracking relationships between entities, as discussed in *Applications*, is the number of entities discovered. 77 million entities were discovered; it is clear that a brute force tactic managing even a single number for every possible pair would not work (requiring 11 petabytes of memory if stored in a short), but managing even just 1,000 numbers for each of 77 million entities would still require 350 gigabytes of memory. Instead, sliding time windows and exponential smoothing are used to make identifying relationships between entities possible.

We find posts on 1.1 million entities per day on average. A popular entity like Hilary Clinton only has 11,000 distinct entities related to her each day, placing a more comfortable upper bound of 12 billion pairings in a time window. If we store summary numbers for entity pairs over the last time window then we can reduce the memory footprint to 23 gigabytes in the worst case. We know that not every entity will have as many pairings on average as Hilary Clinton, and so the footprint in practice is a fraction of that.

Using these results we are able to identify the criteria necessary for extrapolating emerging relationships and identifying meaningful time-series deviations for entities, which helps build the foundation for a framework that allows users to specify what kinds of deviations are considering "interesting".

Future Work

We plan to extend this work in several theoretical and system-implementation directions including:

1. Build the personal information broker that monitors news, Twitter, and social networks in addition to blogs. Help build other example applications such as radiation detection and power grid management.
2. Develop algorithms for generating events on streams of textual data that are efficient in storage space and computational time and which can identify correlations between arbitrary entities.
3. Model a user's context so that information shows up in a user's dashboard, particularly information associated with an auditory or visual alert, is appropriate for the user's context at that point in time.
4. Develop analytic models and carry out experiments to determine the optimum mapping of the communication model to parallel hardware.

Summary

This paper shows that federated event-driven systems help in fusing heterogeneous data from multiple organizations. The architecture integrates SOA and EDA. We demonstrate the advantages of event processing languages (EPLs) that are multilingual compositions of notations appropriate for different actions. We show why the hardware platform suitable for implementing federated event-driven applications is a cluster of message-passing computers each of which consists of multicore processors. The paper makes the case by considering a collection of example applications — the personal information broker (PIB), radiation detection, and power grid management — and evaluating a prototype implementation of a component of the PIB.

A dataflow graph is the meta-specification for composing components specified in different notations. The dataflow graph is mapped to a computational graph where nodes may invoke services and carry out other computational steps, and communicate by sending objects or messages containing serialized objects to queues of other processes. The computational graph is then mapped to a parallel hardware substrate consisting of clusters of message-passing computers each of which consists of multicore processors.

References

Branson, M.; Douglis, F.; Fawcett, B.; Liu, Z.; Riabov, A.; and Ye, F. 2007. CLASP: Collaborating, Autonomous Stream Processing Systems. *Lecture Notes in Computer Science* 4834:348.

Gawlick, D. 2001. Infrastructure for Web-Based Application Integration. In *Proceedings of the 17th International Conference on Data Engineering*, 473–476. IEEE Computer Society Press; 1998.

Kahn, G. 1974. The semantics of a simple language for parallel programming. *Information Processing* 74:471–475.

Misra, J., and Chandy, K. 1981. Proofs of networks of processes. *IEEE Transactions on Software Engineering* 7(4):417–426.

Paschke, A. 2006. ECA-LP / ECA-RuleML: A Homogeneous Event-Condition-Action Logic Programming Language. In *Proceedings of the Int. Conf. on Rules and Rule Markup Languages for the Semantic Web (RuleML06)*.

von Ahn, L.; Ginosar, S.; Kedia, M.; Liu, R.; and Blum, M. 2006. Improving accessibility of the web with a computer game. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, 79–82. ACM New York, NY, USA.