Title: Cacophony: Building a Resilient Internet of Things

Author: John Brock

Affiliation: Department of Computer Science University of California, Irvine, Irvine, CA, USA

Email: jhbrock@ics.uci.edu

Contact Author: Donald J. Patterson

Affiliation: Department of Informatics, Department of Computer Science University of California, Irvine, Irvine, CA, USA

Email: djp3@ics.uci.edu

Abstract:

The proliferation of sensors in the world has created increased opportunities for context-aware applications. However, it is often cumbersome to capitalize on these opportunities due to the difficulties inherent in collecting, fusing, and reasoning with data from a heterogeneous set of distributed sensors. The fabric that connects sensors lacks resilience and fault tolerance in the face of infrastructure intermittency. To address these difficulties, we introduce Cacophony, a network of peer-to-peer nodes (CNodes), where each node provides real-time predictions of a specified set of sensor data. The predictions from each of the Cacophony prediction nodes can be used by any application with access to the web. Creating a new CNode involves three steps: (1) Developers and domain-knowledge experts, via a simple web UI, specify which sensor data they care about-possible

sources of sensor data include stationary sensors, mobile sensors, and the real-time web. (2) The CNode automatically aggregates data from the relevant sensors in real time using a JXTA-based peer-to-peer network. (3) The CNode uses the aggregated data to train a prediction model via the Weka machine-learning library (Hall, 2009). Real-time predictions made by the CNode are then made publicly available to applications that wish to use data from a CNode’s particular set of sensors. The real-time predictions themselves can also be used recursively as sensor data, enabling the creation of CNodes that make predictions based on other CNodes.

1 Introduction

Advancements in manufacturing efficiencies and the subsequent reduction in prices of sensors have caused their deployment to rapidly increase. From barometers on cell-phones and motion sensors on light switches in offices, to seismographs in civil infrastructure and smart meters in homes, physical sensors are proliferating. In order to facilitate system maintenance and the analysis of data, many of these sensors are connected to apps and to Internet services. At the same time, software, both simple and sophisticated, is being used to create “virtual” sensors; for example, geographic sentiment analysis derived from Twitter feeds that capture the mood of a city (Kouloumpis, Wilson, Moore, 2011), and search engine aggregations that identify disease outbreaks (Ginsberg, 2009). While the spread of sensors has created increased opportunities for context-aware applications and is tightly coupled with the Internet of Things (Atzori, Iera, Morabito,2010), there is little accommodation for redundancy in the face of sensor failure or disconnection. As a result, it is difficult to use the opportunities afforded by the scale of this trend when also faced with a future of intermittent infrastructures. There are difficulties inherent in discovering, collecting, fusing, and reasoning with data from the heterogeneous set of distributed infrastructures. Statistical machine learning offers a rich set of techniques for reasoning from large amounts of data, such as that provided by ubiquitous sensors. However, applying these techniques can be difficult because understanding the nuances of machine learning is difficult (Domingos, 2012). Domain expertise is essential to most successful machine learning projects because an expert’s experience and insight is critical to choosing which algorithms to use, which data should and should not form the input to the statistical model, and how the structure of that data and the learning algorithms interrelate. This is assuming that you can even find the relevant data in a form that is amenable to automatic processing. Unfortunately, this expertise is needed over and over again because there is not currently an easy way to make the details of successful models publicly available in a ready-to-use form. There is no large, dedicated public catalogue of trained predictive models from which application developers or researchers can pick-and-choose for their own applications, or upon which they can easily improve. This duplication of effort causes the effective use of sensors to lag far behind their deployment as duplicate teams “reinvent the wheel”. In the face of these trends and challenges, we introduce Cacophony, a network in which computational processes, or “nodes”, provide value-added services for networked physical and virtual sensors. Cacophony exists on a peer-to-peer network to provide resiliency against single-point failures that would otherwise prevent access to underlying sensors and provides services to substitute for the sensor if it becomes completely unreachable. Such services include:

- Predictions: Based on historical observations of sensors and concurrent observations of related sensors (“features” in machine learning parlance), future sensor values can be estimated.

- Robustness: In the presence of sensor failure or sensor in-accessibility, Cacophony can provide current estimates of a sensor’s values based on historical observations and related sensors.

- Transparency: By asking for the configuration of a node, anyone can evaluate the basis on which predictions are made.

- Improvement: By minimally modifying the configuration of an existing node and then using it as the basis for a new node, predictive models can accommodate new domain knowledge and new input sensors as features. This can be done with less expertise than initially creating the model may have required.

- Homogeneity: By creating a consistent interface to sensors on the Internet of Things, Cacophony nodes support developers in the creation of applications; applications only need to implement a single connection to the Cacophony network in order to find any relevant sensor data.

- Scaling: By virtue of using a decentralized peer-to-peer approach, Cacophony supports extremely large resilient networks of sensors.

In this paper, we give an overview of the Cacophony architecture and user interface, followed by descriptions of two context-aware applications that leverage Cacophony.

2 Network Architectures

At a high-level, Cacophony is a peer-to-peer network of nodes and a directory for discovering these nodes. A Cacophony prediction node (CNode) is a predictor for some sensor value known as the “target”. Each node regularly retrieves information (known as “features”) from some set of sensors, and uses that information to create a learning model for predicting the target. Ground truth is obtained by reading text values from a web page or RESTful web service. The learning model is made available via the CNode’s web interface, both on a webpage and through a REST API. In addition to the CNodes, the Cacophony network includes at least one directory node. This directory node provides a public list of the known CNodes, along with relevant information, such as what the CNode is predicting and what sensors it is using to make predictions. This list helps users determine if a certain value is already being predicted, or if they need to make a new CNode to predict the desired value.

2.1 Creating and Configuring a New CNode

Launching a CNode process creates new CNodes. A new node can be configured de novo, or it can be instructed to copy the configuration of an existing CNode. In either case, configuration can be performed through REST API calls to the running node or via a web interface provided by the node.

If the CNode is being configured de novo via the web interface (i.e., it is not copying the configuration of an existing node), a user first finds a source containing the ground truth for whatever values he or she wants to predict (for example, if the user wants to predict the temperature, he or she must find a webpage that is regularly updated with the temperature). Then, on the CNode’s configuration webpage, the user inputs the URL of the ground truth webpage. This will load the ground truth webpage inside the CNode configuration webpage. Once the ground truth page has loaded, the user simply clicks on the desired piece of data, and the CNode records the XPath of that element, i.e., the element’s location in the Document Object Model (DOM). The user then selects other sources of data from a list of existing CNodes provided by the Cacophony directory as features from which the model is trained.

The XPath specifies a text element, but it is possible that the actual target or feature of interest is a substring of what is found at the given XPath. Therefore, if the XPath alone is not precise enough to extract the desired target or feature, the user can supplement it with a regular expression.

If a new CNode is not being configured from scratch, the user can initialize it with the configuration of an existing CNode. The user simply needs to specify the address of the existing node within the peer-to-peer network of CNodes, and the new node will automatically retrieve the appropriate configuration information. This would be a way to rapidly create a sensor that is similar to another existing sensor, or to incorporate changes into the design of an existing sensor in an effort to get better statistical performance.

2.2 Network Protocols

Communication within the Cacophony network is accomplished via p2p4java[1], which is a modification of JXTA (an open source peer-to-peer protocol originally developed by Sun Microsystems). p2p4java offers several advantages over alternative network protocols. For example, consider a smartphone on some wireless carrier’s network: we wish to make this phone’s sensor readings available to a CNode. One naïve approach is to run a web server on the phone that provides the sensor values via a REST API. However, wireless carriers typically use firewalls and network address translation (NAT), which can make it impossible to access this web server from the greater Internet. Fortunately, JXTA offers a solution via relay peers, which enable communication with devices that are behind NATs or firewalls. p2p4java enables this functionality to be extended to Android phones so that a background application on the phone can declare itself a JXTA peer, enabling a CNode to contact it for sensor data.

CNodes can retrieve predictions from other CNodes via the p2p4java network, i.e., a new CNode can use an existing CNode’s predictions as features. In this way, the Cacophony nodes can form a directed graph, where outputs (predictions) from some nodes are used as inputs

1 https://github.com/djp3/p2p4java

(features) for other nodes. JXTA and our p2p4java extensions are open-source, operate on desktop and Android platforms, and feature advanced cryptographic security.

Internally, Cacophony uses the peer-to-peer network for reliability, robustness, and decentralization. However, HTTP is used by the Cacophony network to retrieve sensor data from the rest of the web and to make predictions available to applications that don’t participate in the peer-to-peer network. Each node in the Cacophony network includes a lightweight web server, which serves a webpage for making configuration changes and for displaying status information. The web server also supports a RESTful service for supplying predictions.

2.3 Retrieving Sensor Data and Generating Predictions

On a regular basis, each Cacophony prediction node loads the ground truth’s webpage to check for changes. If the ground truth’s value has changed, the CNode then also retrieves the feature values. Retrieved data is stored in an SQLite database along with all of the previously retrieved values. All of this data is used to train a predictive model via the Weka machine learning library. Whenever a CNode retrieves new sensor data, that node updates its model internally.

Real-time predictions based on the updated model are publicly available to any applications with access to the web, under the Internet of Things SOA model (Atzori, Iera, Morabito, 2010). Notably, the predictions themselves can also be used as sensor data, enabling the creation of CNodes that make predictions based on other CNodes. In order to prevent loops from creating endless fetching of data, basic caching functions are supported that limit the rate at which features are fetched and predictions are calculated.

In addition to just creating a node using the shared configuration of another node, CNodes can be configured to also share their stored data. This is critical for newly created CNodes, since machine learning algorithms can only produce reasonable predictions once a sufficient amount of training data have been accumulated. Sharing data supports rapid bootstrapping of CNodes.

Note that since users are able to clone and modify whichever CNodes are making the best predictions, an iterative evolutionary process results: the most accurate CNode can be observed and then cloned to create multiple new CNodes, and those clones can then be modified incrementally in an attempt to create a more accurate CNode. In principle, this process can be done automatically, but this is still in the realm of future work. Even as a manual process, this enables the accuracy of CNodes to increase over time.

See Figure 1 for a diagram of the Cacophony network.

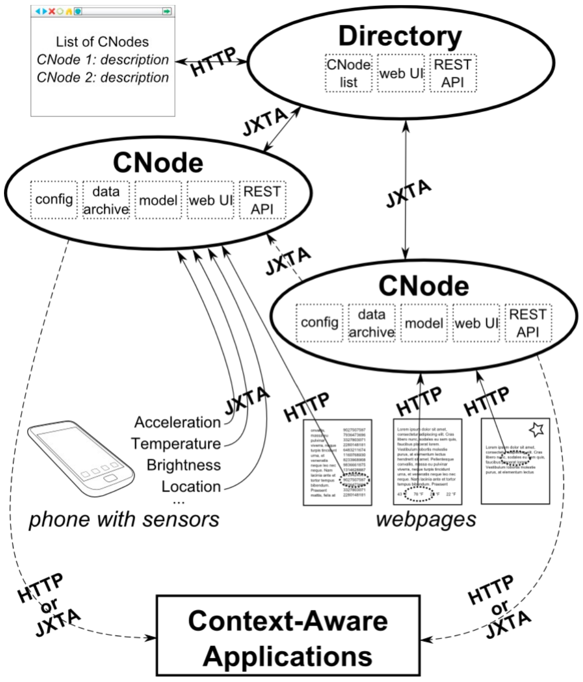
List of CNodes
CNode 1: description
CNode 2: description
HTTP
Directory
CNode list
web UI
REST API
JXTA
JXTA
CNode
config
data archive
model
web UI
REST API
JXTA
CNode
config
data archive
model
web UI
REST API
JXTA
HTTP
HTTP
HTTP
Acceleration
Temperature
Brightness
Location
...
phone with sensors
webpages
HTTP or JXTA
HTTP or JXTA
Context-Aware Applications

Figure 1: A diagram of the Cacophony network showing the flow of sensor data, predictions, and CNode metadata. Dashed lines are predictions being retrieved from a CNode.

3 Related Work

One class of applications that Cacophony was built to support is context-aware applications. Issues of reusability and abstraction when building context-aware applications were previously addressed by Salber with the Context Toolkit (Salber, Dey, Abowd, 1999). Cacophony builds on their ideas by specifying more specific support for machine learning, providing a mechanism for scaling, and developing an implementation that supports modern smartphones.

Previous work by Heer, et al., on the liquid project addressed ways to create a scalable, distributed system for supporting context-aware computing (Heer et al., 2003). Streaming database research inspired their work. Cacophony, in contrast, is designed to function within existing web protocols. In Cacophony, creating a node that collects data from a sensor requires a user to specify how the data can be found. This requires identifying a location on the Internet where the data resides. For example, if the data is embedded in a webpage, a URL and an HTML-path query (or more generally an XPath query) are needed to retrieve the desired data; these parameters need not be entered explicitly by the user, but can be inferred from user interface actions. There is a large body of programming by demonstration (Paynter, 1996) research that is directly relevant to this process of automatically extracting queries in response to user actions. Several systems have already been built that use programming by demonstration to construct screen-scraping information agents. For example, Bauer, et al., built a system to extract text from a webpage based on guidance by the user (Bauer, Dengler, Paul, 2000; Bauer et al., 2000). The system allowed a user to create information agents for retrieving text from webpages, and used programming by demonstration so that the user did not need to manually specify the exact behavior of these information agents. Their system was also robust enough to handle significant changes to a webpage's layout. When the system could not recover from a change to a webpage's layout, it would alert the user, who could then provide new examples to the information agent. The Internet Scrapbook was another system using programming by demonstration to retrieve text from webpages (Sugiura, Koseki, 1998) The user specified which part of a webpage he or she was interested in by copying the text in the desired portion to the Scrapbook page. Text from different webpages could appear on the Scrapbook page, and would reflect updates to the source webpages. Cacophony incorporates these ideas into its configuration (and maintenance) process.

Alba, et al., also describe various ways for retrieving data from the web, such as screen scrapers and APIs (Alba, Bhagwan, Grandison, 2008). When performing screen scraping, there are several ways to refer to part of a webpage, such as the relative positions of elements, regular expressions, and XPath expressions. The Internet Scrapbook used the relative positions of elements, by analyzing the location of headers relative to other text (Sugiura, Koseki, 1998). Regular expressions are a useful way to refer to part of a webpage, but it is easy to under specify or over specify the expression. If the regular expression is too vague, parts of the webpage may be retrieved other than the portion that was desired. If the regular expression is too specific, slight changes in the webpage’s text may result in nothing being retrieved. Fortunately, fairly reliable techniques exist for automatically generating regular expressions based on user input, such as blockwise grouping and alignment (Fernau, 2005). XPath expressions pose problems similar to those posed by regular expressions. XPath expressions can be too vague or else overly specific; specifying the appropriate XPath falls under an area of research known as XPath containment (Vion-Dury, Layaïda, 2003). For simplicity, we rely on XPath expressions to specify a feature on a webpage, and require that the feature be directly inside an HTML element with a unique ID attribute.

Baldauf et al., when discussing various context-aware systems, make a distinction between physical sensors, virtual sensors, and logical sensors (Baldauf, Dustdar, Rosenberg, 2007) Physical sensors are typically hardware sensors that capture simple physical measurements, such as temperature or acceleration. Virtual sensors report data based on software systems; Baldauf and Dustdar provide the example of using event location entries in an electronic calendar system to determine a person’s location (as opposed to using a physical sensor such as a GPS device). Logical sensors provide data based on a combination of data from physical and virtual sensors. Cacophony can consume information from all three types of sensors as long as the sensor’s output is available via an HTTP request. Predictions made by Cacophony can be further built upon as they themselves form virtual sensors and/or logical sensors.

4 User Interface

The CNodes themselves, once they are configured and running, require no UI for collecting and processing data to generate predictions. Cacophony provides a “directory” user-interface tool as a means for people to search for CNodes, to ensure their CNodes are up and running, and to discover CNodes whose sensor data might serve as inputs for their own CNodes’ predictions. To accomplish these tasks one needs to know what CNodes exist and what data they provide.

4.1 Methodology

The directory is itself a peer on the p2p4java network. The directory collects information, both passively and actively, about CNodes in the network. As a passive collector, the directory receives information from CNodes that optionally and periodically announce their existence. Each CNode has a default Cacophony directory with which it can register, but in general, anyone can operate his or her own directory. The directory compares the name of any registering CNode against a list of known CNodes. If no match is found, the directory queues the CNode name in order to retrieve additional information when resources are available. Similar to a web-crawler, the directory visits each CNode in this queue and sends a message requesting the CNode’s configuration. The directory parses and indexes all relevant information from this configuration file. Of special note are any CNodes that serve as features, as these extend the frontier of CNodes known by the directory. Any unknown newly discovered CNodes are added to a queue for future crawling. Since CNodes are not required to register themselves and any attempts to register are “best effort”, this active crawling serves as a compliment to passively waiting for registrations. Finally, known CNodes are revisited periodically to retrieve any updates to their configuration or dependencies.

4.2 Collection of CNode Properties

We categorize the CNodes based on the configuration data retrieved from the CNode registration and crawling. This includes its name on the JXTA network, its features, and metadata about the CNode. Predictions have an associated Java data type (from Weka) and an optional qualitative description. Additionally, the metadata includes machine learning algorithm selections, associated options, guidance on polling frequency, database size and information about node health. Predictions can represent a wide variety of data for example: network throughput, in bytes, for a border router at 137.164.24.49; a barometric reading, in mbars, from 33.684167 to -117.7925; blood sugar levels, in mmol/L, of John Smith; or the NASDAQ Composite index in dollars. The relationships between CNodes are recorded as the flows of predictions from one CNode to another.

4.3 Visualizing the Cacophony Network

Once the directory node has collected information about CNodes, it can provide a visualization showing interrelationships between CNodes and the classification of each CNode. The CNode network is visualized as a directed graph, where a node represents each CNode, and an information flow from one CNode to another is represented as a directed edge. The visualization enables one to quickly understand CNode relationships, to gauge the complexity of the network, and to recognize potential configuration errors.

4.4 Finding CNodes

A CNode may be found through a variety of means. The CNode representations can be directly manipulated via the UI or searched for by address structure or attribute. The address structure of a CNode is a URL in the form of p2p://<device_name>/<semanticpath>/<cnode>, entering the device name will highlight all CNodes contained therein or entering the full address will display the CNode itself. Any of the attributes described previously can be searched in a parametric fashion. All CNodes which match the query are highlighted.

4.5 Interacting with CNodes

When a CNode is selected, various actions are possible: retrieving the current prediction or making future predictions, cloning the CNode, observing current node operational health (such as load and fetching frequency), and displaying configuration attributes. Retrieving the current prediction returns the selected CNode’s most recent prediction. Cloning a CNode outputs a configuration file and the necessary data for making a duplicate CNode. Displaying configuration attributes returns the extended properties not already shown in the UI. Additional features may include retrieving past predictions or retrieving the values used to produce the current prediction.

5 Monitoring Decentralized Infrastructure

One proposed application of Cacophony is to use it to monitor alternative decentralized infrastructures (ADIs). Small-scale attempts to create alternative infrastructures (food, energy, water, etc.) enjoy few of the economies of scale of centralized infrastructure (e.g., rooftop gardening vs. agribusiness), but larger efforts typically require more inputs (such as fertilizer and pesticides) that eventually create expensive externalities. Decentralized efforts, however, currently tend to be isolated, inefficient activities that do not address regional, community-based needs. Cacophony could be used to help these ADIs scale up and be operated with intelligent, computer-based management. Resilient, open-source software can be used to help merge these small-scale efforts into a more integrated, regional forms.

Initial prototypes of this software have been developed but substantial work is still needed in order to allow it to monitor real-world alternative infrastructures. What follows are some examples of how

Cacophony could support alternative infrastructures.

Organization of Existing Sensors: This project does not seek to deploy new sensors, but to integrate existing sensors into a network. In the pairing process of a Cacophony node with an underlying sensor, the sensor is exposed to the rest of the network with a consistent data interface (e.g., API). The node/sensor pairs become searchable through a directory and the complexity of gathering data from now-exposed sensors is greatly reduced. Examples of sensors that we have targeted include smart-meters on homes, electric current sensors on solar panels, irrigation system operational status, micro-weather stations, and smart-phone/wearable sensors.

Intermittency Tolerance: The CNodes communicate using a peer-to-peer networking protocol that is itself resilient to intermittency, but also, as a result, supports continued monitoring in the face of partial infrastructure outages.

Scaling: By using a decentralized peer-to-peer network, the CNodes support extremely large networks of sensors. We are designing based on incorporating approximately 10,000 sensor values into our deployment, many focused on geographic specializations.

Analytics: By incorporating statistical modeling of the sensor values, CNodes can be given predictive analytics that enable them to function on par with utility-grade infrastructures.

Operator Reflection: Consistent, reliable access to decentralized sensor systems coupled with high quality user-interfaces will enable alternative infrastructure owner/operators insight into how to manage their ADIs more effectively.

Research Data: Through monitoring of sensors, we will be able to collect empirical data that can influence future designs.

Applications that are built on top of Cacophony will be able to harvest information from a network of sensors-including physical stationary sensors, mobile and worn sensors, and virtual or physical social sensors and will provide richer insights and potentially support novel use cases.

6 Conclusions

In this paper we have introduced the architecture and vision for Cacophony, a resilient distributed machine-learning layer for the Internet of Things. Cacophony provides value-added services to existing sensors that are accessible on the web. These services include system architecture enhancements such as sensor robustness, interface homogeneity, and scalability. They also include data-oriented services such as historical sensor logging, statistical analysis of data, and predictive estimates of future readings.

An important administrative tool that Cacophony supports is a directory that functions like a web search engine for Cacophony-managed sensors. Through this directory, people and applications can discover and monitor what sensor data is available on the Cacophony network. With this knowledge, new services can be created and existing services can be incrementally improved.

We developed Cacophony in response to a practical need to support open-source applications that needed to leverage wider portfolios of sensor data than were previously available. For example, previous research supported the automatic inference of context descriptions from the sensors on a single device. Cacophony has allowed us to write applications that determine the context of an infrastructure from a wide range of sensors that span many devices, administrative domains, and intended uses.

In the future we hope to provide more support to users who would like to create their own CNodes in the Cacophony network. This includes automatic inferring of features for target predictions, user interface enhancements for launching CNodes, and broader deployments in support of a variety of applications. Network effects make Cacophony increasingly valuable as the number of sensors available on the Cacophony network increases.

7 Acknowledgments

Portions of this work were funded by grants from the University of California, Irvine’s Center for Organizational Research.

References

A. Alba, V. Bhagwan, and T. Grandison, 2008. “Accessing the deep web: when good ideas go bad”. In: *Companion to the 23rd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2008, October 19-13, 2007, Nashville, TN, USA*. Ed. by Gail E. Harris. ACM, 2008, pp. 815–818. at http://doi.acm.org/10.1145/1449814.1449871

L. Atzori, A. Iera, and G. Morabito, 2010. “The Internet of Things: A survey”. In: *Comput. Netw.* 54 (15 2010), pp. 2787–2805. ISSN: 1389-1286. at http://dx.doi.org/10.1016/j.comnet.2010.05.010

M. Baldauf, S. Dustdar, and F. Rosenberg, 2007. “A survey on context-aware systems” In: *IJAHUC* 2.4 (2007), pp. 263–277.

M. Bauer, D. Dengler, and G. Paul, 2000. “Instructible information agents for Web mining”. In: *IUI*. 2000, pp. 21–28 at http://doi.acm.org/10.1145/325737.325758

M. Bauer et al., 2000. “Programming by Demonstration for Information Agents”. In: *Commun. ACM* 43.3 (2000), pp. 98– 10 at http://doi.acm.org/10.1145/330534.330547

P. Domingos, 2012. “A few useful things to know about machine learning”. In: *Commun. ACM* 55.10 (Oct. 2012), pp. 78– 87 at http://doi.acm.org/10.1145/2347736.2347755

H. Fernau, 2005. “Algorithms for Learning Regular Expres- sions”. In: *Algorithmic Learning Theory, 16th International Conference, ALT 2005, Singapore, October 8-11, 2005, Pro- ceedings*. Ed. by Sanjay Jain, Hans-Ulrich Simon, and Et- suji Tomita. Vol. 3734. Lecture Notes in Computer

Science. Springer, 2005, pp. 297–311 At http://dx.doi.org/10.1007/11564089_24

J. Ginsberg et al., 2009. “Detecting influenza epidemics using search engine query data”. In: *Nature* 457.7232 (Feb. 2009), pp. 1012–1014. URL: http://dx.doi.org/10.1038/nature07634

M. Hall et al.,2009. “The WEKA data mining software: an up- date”. In: *SIGKDD Explor. Newsl.* 11 (1 2009), pp. 10–18 at http://doi.acm.org/10.1145/1656274.1656278

J. Heer et al.,2003. “liquid: Context-Aware Distributed Queries”. In: *Ubicomp*. Ed. by Anind K. Dey, Albrecht Schmidt, and Joseph F. McCarthy. Vol. 2864. Lecture Notes in Computer Science. Springer, 2003, pp. 140–148 at http://dx.doi.org/10.1007/978-3-540-39653-6_11

E. Kouloumpis, T. Wilson, and J Moore, 2011. “Twitter sentiment analysis: The good the bad and the omg!” In: *ICWSM* 11 (2011), pp. 538–541 at https://www.aaai.org/ocs/index.php/ICWSM/ICWSM11/paper/view/2857

G.W. Paynter, 1996. “Generalising programming by demonstration”. In: *Computer-Human Interaction, 1996. Proceedings., Sixth Australian Conference on*. 1996, pp. 344–345 at http://dx.doi.org/10.1109/OZCHI.1996.560161

D. Salber, A. K. Dey, and G. D. Abowd, 1999. “The context toolkit: aiding the development of context-enabled applications”. In: *CHI*. Ed. by Marian G. Williams and Mark W. Altom. Pittsburgh, Pennsylvania, United States: ACM Press, 1999, pp. 434–441 at http://doi.acm.org/10.1145/302979.303126

A. Sugiura and Y. Koseki, 1998. “Internet Scrapbook: Automating Web Browsing Tasks by Demonstration”. In: *ACM Symposium on User Interface Software and Technology*. 1998, pp. 9–18 at http://doi.acm.org/10.1145/288392.288395

J-Y. Vion-Dury and N. Layaïda, 2003. “Containment of XPath expressions: an inference and rewriting based approach”.In: *Proceedings of the Extreme Markup Languages, 2003 Conference, 4-8 August 2003, Montréal, Quebec, Canada* at http://www.mulberrytech.com/Extreme/Proceedings/html/2003/Vion-Dury01/EML2003Vion-Dury01.html