Research Abstract for Semantic Anomaly Detection in Dynamic Data Feeds with Incomplete Specifications

Orna Raz
School of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213 USA
orna.raz@cs.cmu.edu

1. Motivation

Much of the software we use for everyday purposes incorporates elements developed and maintained by someone other than the developer. These elements include not only code and databases but also dynamic data feeds from online data sources. Although everyday software is not mission critical, it must be dependable enough for practical use. This is limited by the dependability of the incorporated elements.

It is particularly difficult to evaluate the dependability of dynamic data feeds, because they may be changed by their proprietors as they are used. Further, the specifications of these data feeds are often even sketchier than the specifications of software components.

We propose a method and tools for inferring invariants about the normal behavior of dynamic data feeds. We use these invariants as proxies for missing specifications to perform on-going detection of anomalies in the data feed. We concentrate on *semantic anomaly detection*: identifying occasions when a dynamic data feed is delivering unreasonable values, even though its behavior may be superficially acceptable (i.e., it is delivering parsable results in a timely fashion). Initial results demonstrate the feasibility of our approach [28].

2. Thesis statement

We can infer useful characteristics of the normal behavior of a dynamic data feed and use these characteristics as proxies for missing specifications, to augment any existing specifications.

These augmented specifications suffice for practical semantic anomaly detection. Their inference can be done automatically, to a large extent.

3. Approach

Our long-term goal is to improve the dependability of everyday software systems. We are especially interested in systems that rely on elements that remain under the control of the elements' proprietors. An example of such elements, which are also in particular need of increased dependability, is dynamic data feeds from online data sources. Examples of data sources include stock quotes, weather forecasts, airline ticket prices, and news reports. A data feed captures a particular usage of a data source (Figure 1). Examples of data feeds from the above data sources include stock quotes for a specific company, the weather forecast for a specific city, airfare for specific origin and destination, and news for a specific topic.

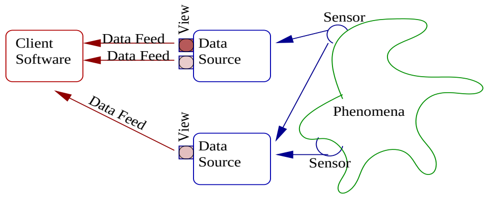

Figure 1: A client software uses a dynamic data feed that results from a view (query/filter) on content provided by a data source. A data source samples phenomena via sensors.

On the one hand, timely availability greatly increases the usefulness of dynamic data feeds and hence their value. On the other hand, it is hard to detect anomalies in such data feeds because of their incomplete specifications. Although anomalies in a data feed indicate anomalies in the data source providing this data feed, in our setting the system developer does not have control over the data source, so dependability enhancement can only be done by the client of the data. For the client, the data source is represented by the data feeds the client uses. We, therefore, concentrate on data feeds. We further restrict our domain to data feeds that supply well structured content that consists of numeric (and possibly categorical) attributes [1].

We are interested in semantic failures. Such failures are hard to detect without good (relevant and correct) specifications of the underlying semantics, because a system using the faulty data feed might superficially appear to behave well, yet the service this system delivers should not be relied upon. This is the availability facet of dependability [9], under a semantic fault model: the readiness for usage of a data feed that delivers parsable results, as indicated by whether

[1] Free text raises additional challenges, which are beyond the scope of this work

or not it delivers reasonable results. To measure and assess this availability, the delivery of semantically correct service needs to be estimated, with respect to the alternation of semantically correct and semantically incorrect service.

Detecting semantic failures would enable us to estimate the semantic correctness of a service, because we would be able to distinguish between correct and incorrect service. We concentrate on anomaly detection because the incomplete specifications of data feeds do not enable us to distinguish a failure from other anomalous behavior that does not constitute a failure.

Fault tolerance approaches to detection often use a state-space method [32]. This requires specifications of normal, degraded, and abnormal states, as well as transition probabilities between states. Masking, which does not require detection, requires specifications of outputs and their selection.

However, a particular problem in the domain of dynamic data feeds is that their specifications are sketchy and incomplete. Such specifications are not good enough to model the semantics of the data feeds and, therefore, not good enough to support anomaly detection that relies on specifications. In [29] we noted state-space models are difficult to work with when the specifications are inaccurate and suggested an alternative incremental-improvement, gradient view.

Our approach to anomaly detection follows the gradient view, overcoming the limitation of requiring precise definitions of states and transitions. We suggest to *infer* characteristics of the data feed, using and adapting existing statistical and machine learning techniques for the inference. We assume the data feed usually behaves normally. The inferred characteristics, which we use as proxies for missing specifications, can augment existing specifications and domain knowledge. These characteristics are captured in the form of *invariants*. We concentrate on boolean and statistical invariants. *Anomalies* are detected when an invariant is evaluated to false over fresh observations (observations that were not used for inferring the invariant).

The main research challenge is to infer invariants that are strong enough to separate normal behavior from abnormal behavior. This entails additional research challenges, related to using and adapting existing techniques for invariant inference: model selection for a specific technique and data feed (tuning parameters of a techniques and preprocessing the data), choosing the size and period of time of the training data, and choosing a collection of techniques that is good-enough for inferring invariants for cost-effective semantic anomaly detection.

3.1 Major steps

We distinguish between the setup and the usage stages of our invariant inference framework. The frequency with which these stages are executed differs greatly and therefore so can the automation level: some tools we supply may only be semi-automated (e.g., procedures, heuristics; see discussion in Section 5.2).

Sections 3.1.1 and 3.1.2 describe the steps involved in the setup and usage stages, respectively. For each of the steps we describe, our goal is to supply tools to aid in performing the specified task. We also specify how certain domain knowledge and specifications (existing knowledge) could be used in each of these steps.

3.1.1 Setup stage

The setup stage can be viewed as a gray-box into which the user enters a data feed and from which the user gets a suite of invariant inference techniques, tuned for the given data feed. Existing knowledge could be used in each of the steps to direct selection and tuning of techniques, thus potentially improve the resulting suite. Creating the suite of techniques includes the following steps:

1. Select "raw" techniques: a set of techniques that may be applicable to the data feed. We want a variety of techniques, in terms of the invariants they infer.
   Domain knowledge and specifications may be used to direct the selection of techniques.
   Tool support will be needed for matching data and techniques.

2. Prepare the data and the techniques for trial. This includes: augmenting a technique (e.g., augmentation for noise handling); preprocessing the data (e.g., binning, dividing into week vs. weekend days).
   Domain knowledge may be used to direct preprocessing.
   Tool support will be needed for augmentation and for pre-processing.

3. For each of the techniques of step 1 run trials for model selection, using as a criterion the strength of the inferred invariants for semantic anomaly detection. This includes: recommending the amount and period of time of the data, including adapting to changes (handling concept drift); setting values of tunable parameters of the technique.
   Domain knowledge and specifications may be used for setting the amount and period of data and the values of tunable parameters of a technique.
   Tool support will be needed for determining amount of training data and for tuning technique parameters.

4. Turn the individual experiments into a suite.
   Domain knowledge and specifications can be incorporated as output from an additional technique.
   Tool support will be needed for technique selection.

3.1.2 Usage stage

Each of the techniques in the suite produced by the gray-box described in Section 3.1.1 is used for invariant inference, to produce proxies for missing specifications. In the work we propose we use the specification proxies for anomaly detection: they define a model for anomalies. The usage stage consists of the following steps:

1. Infer an expectation over a moving window of observations: use each of the techniques in the suite to infer invariants over observations in a training set (some number of observations from the data feed that is used for training)

2. Detect anomalies: evaluate the invariants in the expectation over fresh observations. These observations lie ahead of the window used for invariant inference (i.e., future observations). Report an anomaly when an invariant evaluates to false

This approach is intended to work even if no additional information is available. However, existing knowledge could be used and should enable better anomaly detection. Existing knowledge may be used to add invariants to check during the anomaly detection. Tool support will be needed for anomaly detection.

4. Expected contribution

We expect to

- provide a method and tools for inferring characteristics of normal behavior of a dynamic data feed that can be used as proxies for missing specifications,
- demonstrate the usefulness of these characteristics for semantic anomaly detection, and
- qualitatively demonstrate cost-effectiveness.

An additional possible contribution would be to define a model for characterizing techniques and data feeds in a way that is useful for capturing their semantics.

4.1 Possible usages of invariants that characterize usual behavior

We use the inferred invariants for semantic anomaly detection, as it is a first step towards increasing the dependability of dynamic data feeds. However, the inferred invariants may have additional usages, such as: assessing independence of data feeds, helping to deduce likely specifications, and detecting mismatches between specifications and actual behavior. Moreover, our approach for inferring invariants may be useful not only for consumers but also for producers of data feeds. For example, historical behavior within a context may be used to automatically establish sanity checks for entering data into a database.

5. Validation

We map our thesis statement (Section 2) to validation criteria. We need to address three major criteria: breadth, cost, and strength/benefit. We discuss these criteria in Sections 5.1, 5.2, and 5.3, respectively.

5.1 Breadth

Our experimental plan aims to ensure this research explores representing dynamic data feed examples, such as: a single data feed and multiple data feeds with a semantic relation among them. The semantic relation may be redundancy (e.g., quotes for the same stock from different data sources) or correlation (e.g., river level and precipitation).

5.2 Cost

We expect the main cost factor to be human attention: the amount of human intervention, its nature, and the amount of false positives.

For each step in our approach, we will make sure the frequency of using this step matches the level of human attention required in this step. Steps that are infrequent may require intensive human intervention, whereas more frequent steps should be less demanding of human attention. For example, adapting and adding an existing technique to the gray-box is done once. Therefore, is will be satisfactory if we provide a procedure (that may be human attention intensive) for doing so. Inferring invariants is done once every training widow size, therefore, it should be made routine. Detecting anomalies over newly observed data is done very frequently (as frequently as every observation), therefore, it should be automated.

5.3 Strength/benefit

Anomaly detection is not only a first step towards increasing dependability but also the means we use to validate the strength of the inferred invariants: we measure how effective these invariants are for semantic anomaly detection. We use classification accuracy to quantify the effectiveness for anomaly detection.

For each invariant inference technique we want to validate the model selected is effective. We do this by measuring the classification accuracy of the technique, where the baseline of all anomalies to detect consists only of anomalies that could be detected using the kind of invariants the technique is able to infer. In general, we do not expect to get perfect detection, as it is often required to tune parameters for a technique and to pre-process the data.

For a collection of invariant inference techniques we want to validate its detection effectiveness over all the anomalies in the data. We cannot, in general, determine precisely what the set of all anomalies in the data is: our problem is similar to the Kolmogorov complexity [12] so it is undecidable (Kolmogorov complexity has been reduced to the halting problem).

However, our goal is to improve the current situation (humans occasionally look at the data): we settle for anomaly detection that is comparable to what an "ideal" (able to pay attention) human would find. We are surveying the human factors literature to establish a set of baseline invariants, based on human capabilities. If the techniques we use find additional semantic anomalies we will extend this baseline.

6. Related work

Efforts to increase dependability include approaches for preventing problems and approaces for detecting/mitigating problems. Section 6.1 presents work related to prevention. Section 6.2 presents work related to detection/mitigation. Our approach concentrates on detecting semantic problems by the client of a data feed.

6.1 Preventing problems

We describe approaches that address preventing connectivity, syntax/from, and semantic problems in online data sources. In general, prevention and detection/mitigation are complementary, as it is rarely possible to completely prevent problems.

Our approach deals with the situation as it is today. The Semantic Web [6, 11] is a grand vision for a comprehensive solution to syntax/form and semantic failures that requires many additions to the current Web.

Our work is concerned with data quality. Our emphasis is on measuring and increasing the quality of the data by the consumer of this data. Most of the research related to data quality is concerned with measuring and increasing data quality by the producer of the data: providing data quality dimensions for reasoning about data quality in the design of information systems [33], identifying the importance of data consumers when defining data quality [31], and discussing different perspectives of data quality for information products and businesses [24].

Web Services promote the same new development paradigm we do: building applications using as elements services available on the Web. Web Services use XML-based protocols that support discovery of services and automatic information exchange and integration. This is mainly related to connectivity and syntax/form failures, addressing the interface problem. Examples of Web Services include Microsoft's .NET, IBM's dynamic e-business (e.g., WebSphere Application Server, Web services Toolkit), and Oracle's dynamic services.

6.2 Detecting/Mitigating problems

We describe approaches that detect and mitigate connectivity, syntax/form, and semantic problems in online data sources. There are many existing solutions to connectivity and syntax/form problems, but not to semantic problems. For improved dependability, solutions to all types of problems are necessary.

Solutions to individual types of connectivity faults exist, in the form of redundant servers, caching, and archiving. Examples of tools providing such solutions are the Go!Zilla download manager [17], Alexa's archive of pages [7], and Google's caching [16].

Examples of solutions to individual types of syntax/form faults are wrapper induction (a program for extracting data), wrapper re-induction (detecting problems in wrappers and adjusting to changes), and languages that facilitate information exchange between data sources (and support automatic parsing). Wrapper induction and re-induction learn the data prototype from labeled examples and require some domain specific information. Ariadne [21] and TrIAs [10] are examples of research prototypes for wrapper induction and re-induction. RAPTURE [22] is a research prototype for wrapper validation. Kangaroo [20] is a commercial tool that includes help in extracting information from HTML. Languages that facilitate information exchange require well defined semantic and syntactic standards. XML is an example of a language facilitating information exchange between data sources. The Information Manifold research prototype [26] is an example of a system that provides access to a collection of closely related data sources by using a language to describe their contents and capabilities.

The most relevant work to semantic failures seems to be wrapper re-induction and wrapper validation [25, 10, 22], but these often require labeled data and domain specific information. We seek more general solutions.

Data scrubbing (aka data cleansing) is often concerned with enterprise customer data. As a result, tools for data scrubbing are often domain specific: for example, tools for correcting, standardizing and verifying address elements [2]. A main concern in data cleansing is data integration. Tools often have a component of data discovery, that addresses a syntax/form problem (parsing records that have varying formats) [5]. There are also data cleansing products that operate according to a set of rules [1]. Potter's wheel [27] is an interactive data cleaning system that includes transformations (addressing syntax/form problems) and a discrepancy detector. The users need to define domains and algorithms to enforce domain constraints for the discrepancy detector. The system automatically infers appropriate structure for columns in the data and applies the appropriate domain algorithm to detect anomalies. Our work concentrates on general (domain independent) algorithms, but we also recognize the need to support domain knowledge. GritBot [18] provides a domain independent technique for finding anomalies in a data set, to prepare it for data mining. However, GritBot provides one technique only, that is applicable to a single non-dynamic set of data with numeric and categorical attributes. This technique could be incorporated in our approach, as one of the techniques for invariant inference, possibly with little or no adaptation.

Our long term vision is to increase the dependability of software systems through self healing [30]. The work of Forrest et al. [19] has a similar vision, as does the "autonomic computing" initiative of IBM [3]. Forrest et al. concentrate on comparing sequences of events as the basic detection method, and have demonstrated it for the security domain (intrusion detection). The most important step in their approach is a domain specific encoding of a characteristic to monitor. IBM's eLiza [4] provides self-managing servers, dealing mostly with connectivity failures.

6.2.1 Using behavior to characterize properties

Our approach of inferring the characteristics of a data feed from its behavior is similar to work in the areas of program analysis [15, 13, 8] and intrusion detection [23].

The idea of inferring invariants from actual behavior is similar to dynamically discovering likely program invariants [15, 14]. However, the domain, emphasis, and goal of our approach are different. We deal with online data sources rather than programs. We use off-the-shelf techniques rather than provide a new technique for inferring invariants. We aim to increase dependability rather than aid in programming tasks.

"Bugs as deviant behavior" [13] infers beliefs from source code. These beliefs are facts implied by the code. It defines checkers that extract beliefs by instantiating rule templates. Though similar to our approach in the desire to detect incorrectness without knowing what correct is, this approach is static (it requires source code), and is based on principles that are specific to programs (e.g., propagating beliefs, instantiating checker slots). In addition, it only works for a specific set of programmer-defined properties (which is true of any such system).

"Specifications mining" [8] uses a machine learning approach to discover specifications of interface protocols. It uses program executions to extract "scenarios": short frequent interaction patterns. It uses an exiting tool to build a non-deterministic finite state automaton for a scenario. The resulting specifications express temporal and data-dependence relationships among calls to an API and can be formally verified. This approach is based on the observation that common behavior is often correct, thus the specifications mining problem becomes a problem of probabilistic learning from execution traces. This is similar to our approach. However, scenario extraction is specific to programs because it requires scenario seeds in the form of interaction skeletons.

We concentrate on the semantic availability and assume unintentional semantic faults. Intrusion detection concentrates on security and assumes a malicious intruder. Intrusion detection approaches look for behavior that indicates attacks (e.g., network traffic [19]), malicious behavior of a system [19], or malicious users (e.g., by shell commands [23]). In addition, intrusion detection often concentrates on specific sequence-of-events related techniques.

Lane et al. suggest an approach to intrusion detection that includes a personal anomaly detection agent that aids a single user to protect his/her account from abuse [23]. They developed models to learn characteristic sequences of actions generated by the user (the actions they deal with are tokenized UNIX commands). Their focus on a personal assistant is relevant to our work because it raises a set of issues that are relevant to our domain as well: concept drift, online learning, single class learning, and temporal sequence learning [23]. Our work is based on a similar approach: capturing a specific usage (represented by a data feed/personal assistant), then performing anomaly detection by characterizing past behavior, comparing current behavior to this characterization, and detecting an anomaly if the difference is too substantial. However, their work concentrates on a specific domain, uses a specific kind of data, and investigates the best way to apply a specific learning method, whereas we are interested in a more general setting.

7. Acknowledgements

The author thanks her thesis committee members: Mary Shaw, Michael Ernst, Christos Faloutsos, and Philip Koopman.

This research is supported by the National Science Foundation under Grant CCR-0086003 and by the Software Industry Center at Carnegie Mellon University.

References

[1] Datanomic. URL: `http://www.datanomic.com/technology.htm`. Accessed November 2001.

[2] FirstLogic. eDataQuality. URL: `http://www.firstlogic.com/solutions/technologies/iq/edataquality.asp`. Accessed November 2001.

[3] IBM. Autonomic Computing. URL: `http://www.research.ibm.com/autonomic/overview/`. Accessed November 2001.

[4] IBM. eLiza: self-manging servers. URL: `http://www-1.ibm.com/servers/eserver/introducing/eliza/`. Accessed November 2001.

[5] Paladyne data discovery. URL: `http://www.paladyne.com/software/discovery.html`. Accessed November 2001.

[6] World Wide Web Consortium (W3C). The Semantic Web, Activity. URL: `http://www.w3.org/2001/sw/`. Accessed November 2001.

[7] Alexa browser enhancement. URL: `http://www.alexa.com`. Accessed April 2001.

[8] G. Ammons, R. Bodik, and J. Larus. Mining specifications. In *The 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Portland, Oregon, January 2002.

[9] A. Avizienis, J. Laprie, and B. Randell. Fundamental concepts of dependability. Technical report, UCLA CSD Report no. 010028, LAAS Report no. 01-145, Newcastle University Report no. CS-TR-739, 2001.

[10] M. Bauer and D. Dengler. Trias: Trainable information assistants for cooperative problem solving. In *Proceedings of 3rd International Conference on Autonomous Agents*, 1999.

[11] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. Scientific American. URL: `http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html`, May 2001. Accessed November 2001.

[12] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley, 1991.

[13] D. Engler, D. Y. Chen, S. Hallem, A. Chou, and B. Chelf. Bugs as deviant behavior: A general approach to inferring errors in systems code. In *Proceedings of 18th ACM Symposium on Operating Systems Principles*, 2001.

[14] M. Ernst. *Dynamically Discovering Likely Program Invariants*. PhD thesis, University of Washington, Department of Computer Science and Engineering, 2000.

[15] M. Ernst, J. Cockrell, W. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. In *IEEE Transactions on Software Engineering*, pages 99–123, 2000.

[16] Google search engine. URL: `http://www.google.com`. Accessed April 2001.

[17] Go!Zilla download manager. URL: `http://www.gozilla.com`. Accessed April 2001.

[18] Rulequest. GritBot. URL: `http://www.rulequest.com/gritbot-info.html`. Accessed January 2002.

[19] S. Hofmeyr and S. Forrest. Architecture for an artificial immune system. In *Evolutionary Computation Journal*, 2000.

[20] Kangaroo data collection. URL: `http://www.kangaroonet.com`. Accessed August 2001.

[21] C. Knoblock, K. Lerman, S. Minton, and I. Muslea. Accurately and reliably extracting data from the web: A machine learning approach. In *Data Engineering Bulletin*, 1999.

[22] N. Kushmerick. Regression testing for wrapper maintenance. In *Proceedings of AAAI-99*, pages 74–79, Orlando, 1999.

[23] T. Lane and C. E. Brodley. Approaches to online learning and concept drift for user identification in computer security. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pages 259–263, 1998.

[24] Y. W. Lee, P. L. Bowen, J. D. Funk, M. Jarke, S. E. Madnick, and Y. Wand. Data quality in internet time, space, and communities (panel session). In *Proceedings of the 21st International Conference on Information Systems*, December 2000.

[25] K. Lerman and S. Minton. Learning the common structure of data. In *Proceedings of 17th National Conference on Artificial Intelligence*, 2000.

[26] A. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of 22nd International Conference on Very Large Data Bases*, September 1996.

[27] V. Raman and J. M. Hellerstein. Potters wheel: An interactive data cleaning system. In *International Conference on Very Large Data Bases (VLDB)*, Rome, Italy, 2001.

[28] O. Raz, P. Koopman, and M. Shaw. Semantic anomaly detection in online data sources. In *Proceedings of the 24th International Conference on Software Engineering*, May 2002.

[29] O. Raz and M. Shaw. An approach to preserving sufficient correctness in open resource coalitions. In *Proceedings of 10th International Workshop on Software Specification and Design*, pages 159–170, San Diego, California, November 2000.

[30] M. Shaw. Sufficient correctness and homeostasis in open resource coalitions: How much can you trust your software system? In *Proceeding of 4th International Software Architecture Workshop*, May 2000.

[31] D. M. Strong, Y. W. Lee, and R. Y. Wang. Data quality in context. *Communications of the ACM*, 5(40):103–110, May 1997.

[32] A. Villemeur. *Reliability, Availability, Maintainability, and Safety Assessment*. Jon Wiley & Sons, 1992.

[33] Y. Wand and R. Y. Wang. Anchoring data quality dimensions in ontological foundations. *Communications of the ACM*, 39(11):86–95, November 1996.