Enabling automatic adaptation in systems with under-specified elements

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

Philip Koopman
ECE Department
Carnegie Mellon University
koopman@cmu.edu

Mary Shaw
School of Computer Science
Carnegie Mellon University
mary.shaw@cs.cmu.edu

ABSTRACT

Software that people use for everyday purposes is usually not mission critical—some failures can be tolerated. However, this software should be dependable enough for its intended use, even when users change expectations. Software systems that could adapt to accommodate both failures and changing user expectations could significantly improve the dependability of such everyday software. Many adaptation techniques require specifications of proper behavior (for detecting improper behavior) and problem severity, alternatives and their selection (for mitigation and for repair).

However, the specifications of everyday software are usually incomplete and imprecise. This makes it difficult to determine the dependability of the software and even more difficult to adapt.

We address the problem of detecting anomalies—deviations from expected behavior—when specifications of expected behavior are missing. Setting up anomaly detection depends on human participation, yielding predicates that can serve as proxies for missing specifications.

We propose a template mechanism to lower the demands on human attention when setting up detection. We show how this mechanism may be used in our framework for enhancing dynamic data feeds with automatic adaptation. We discuss how the same mechanism may be used in repair. Our emphasis is on detecting semantic anomalies: cases in which the data feed is responsive and delivers well-formed results, but these results are unreasonable.

1. INTRODUCTION

Many systems operate in a closed feedback loop that requires self healing—adapting to failures. Adaptation includes not only adapting to failures but also adapting to changes. Enhancing software we use for everyday purposes so that it could adapt to failures and to changing user expectations could significantly improve the dependability of such software and hence its utility.

Many adaptation techniques rely on detection of improper behavior, mitigation, repair, and possibly on-going maintenance. These activities require knowledge of correct behavior, usually in the form of specifications. Detection requires specifications of a model of proper behavior. Mitigation and repair require specifications of problem severity and of available alternatives and their selection. However, everyday software is often under-specified—it provides only imprecise and incomplete specifications.

The situation is exacerbated when the software incorporates third-party elements such as COTS (Commercial-Off-The-Shelf) components, databases, and dynamic data feeds from online data sources. The latter case is especially difficult, because the proprietor of the data feed may change its semantics, format, or availability while it is being used, and without notice; further, specifications for data feeds are often even sketchier than those for software components. Examples of data feeds include stock quotes for a specific company, airfare for specific origin and destination, prices for a given product, and news for a specific topic.

We address the challenge of cost-effectively enhancing third-party, data-providing elements to support adaptation, when these elements are dynamically changing, outside our control, and under-specified. Dynamic data feeds are an example of such elements, and the one we use in our work.

We concentrate on enabling automatic detection of *semantic anomalies*—cases in which the data provided by the data feed is outside the model of proper behavior. Human intervention is essential in setting up this model because proper behavior is determined by the semantics of the data. Further, proper behavior may also depend on the user and the goal the user has for the data. If the providers of a data feed were to set up the anomaly detection model they would need to specify the semantics of the data feed, all possible usages, and all possible changes to the behavior of the data. This is likely to require much human expertise and time, might produce an incomplete result in spite of the huge investment, and rarely occurs for the everyday elements we are concerned with.

We propose having the users of data feeds set up the anomaly detection. However, to make adaptation practical, the human intervention must be made manageable, both in terms of expertise and of time. We propose a template mechanism to incorporate the human intervention in the adaptation framework. We believe the template mechanism can provide a means for eliciting expectations and generalizations from the user. These are essential for building a model of proper behavior for anomaly detection and for directing repair.

Section 2 presents an example of a data feed and techniques

time	cur	dlow	w52low	beta
10/11/2000 15:30	52.13	48.22	32.53	1.43
10/11/2000 15:40	52.06	48.22	32.53	1.43
...	...	...	...	...
10/17/2000 14:20	54.00	54.00	32.56	1.43
10/17/2000 14:30	53.50	53.44	32.56	1.43

Table 1: Example of a stock quote data feed

we use throughout the paper. Section 3 describes a feasible design of an adaptation framework for dynamic data feeds that is domain independent. Our framework separates steps in which human intervention is required (setting up adaptive invariants for anomaly detection, judging the results of the anomaly detection, providing alternatives and priorities for repair) from steps that could be fully automated (performing anomaly detection). Section 4 explores a mechanism for getting users feedback based on *templates* for adaptive invariants. We believe templates are useful because they could: (1) be employed in all the steps that require human intervention, and (2) reduce demands on human intervention. Section 5 describes other work related to increasing dependability and to making human intervention explicit. Section 6 discusses some open issues and future work.

2. EXAMPLE

We have been developing a method for inferring invariants that can serve as a model of normal behavior of the data feed. This model can then be used to detect anomalies—a critical first step in automatic repair. We present an example of a data feed and existing techniques we use for invariant inference. We use this example throughout the paper.

Table 1 gives an example of a stock quote data feed. The data feed provides information about a specific stock (CSCO) from a specific data source ([8]). A data feed is a time ordered vector of observations. Each observation (a row in Table 1) contains a time stamp and a value for each attribute. This data feed has four attributes: the current value of the stock `cur`, the daily low value of the stock `dlow`, the low value of the stock during the last 52 weeks `w52low`, and a measure of anticipated fluctuations relative to the market fluctuations `beta`.

We have investigated two invariant inference techniques: Daikon [10] and Mean. Daikon was developed for the program analysis domain. It dynamically discovers likely program invariants over program execution traces by checking if pre-defined relations hold. Mean is a statistical method based on estimating a confidence interval for the mean of an attribute distribution. It estimates the mean and standard deviation from the data and expects values to be within a certain number of standard deviations of the mean.

Sections 3 and 4 use this example to illustrate our adaptation framework for dynamic data feeds and our proposed templates for invariants, respectively.

3. ADAPTATION OF DYNAMIC DATA FEEDS

Everyday software must be sufficiently dependable for its intended purpose. Because this software is usually not mission-critical, it may be cost-effective to detect improper behavior

Figure 1: Anomaly detection setup. Requires human intervention

and notify the user or take remedial actions. Detection and repair require specifications. Unfortunately, specifications of everyday software are often incomplete and imprecise. Section 3.1 presents our approach to detection with incomplete specifications. Section 3.2 describes the sorts of repair that may follow detection.

3.1 Detection

In [23] we proposed a method for inferring adaptive invariants about the normal behavior of dynamic data feeds, using and adapting existing techniques for the inference. *Adaptive invariants* are not only static invariants but also relations and values that are expected to change as the behavior of the data changes. We demonstrated it is possible to use these invariants as proxies for missing specifications to perform on-going semantic anomaly detection in the data feed (for a single data feed with numeric valued attributes, in the context of stock market tickers). For the rest of this paper, we use invariants to mean adaptive invariants.

Our invariant inference framework has two major stages: setup and usage. The frequency with which these stages are executed differs greatly and therefore so can the automation level. The setup stage requires human intervention. The usage stage permits human intervention, but does not require it. However, intervention is often desired if repair is to follow.

The setup stage, depicted in Figure 1, 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. Producing the suite includes the following steps: (1) match candidate techniques with the data, (2) if necessary, augment the techniques (e.g., to handle noise) and/or pre-process the data (e.g., make continuous values discrete), (3) find an effective way to use each technique over the data—this requires user intervention (find a training set (moving window) size, find parameter values, select attributes (aka feature extraction)), and (4) determine a good-enough subset of techniques.

For our Section 2 example, the gray-box takes the stock-quote data feed presented in Table 1 as input. The gray-box outputs an augmented Daikon that is capable of handling noisy data, Mean with a parameter set, and for both Daikon and Mean a recommendation for a training set size (the amount of observations to use for invariant inference).

The usage stage, depicted in Figure 2, repeatedly infers invariants and detects anomalies over a moving window of observations over the data feed. First, each of the techniques in the suite is used for invariant inference. Then, each invariant is evaluated over the moving window data. An anomaly is detected when an invariant is evaluated to false.

For our example, the anomaly detector uses each of augmented Daikon and Mean to repeatedly infer invariants over

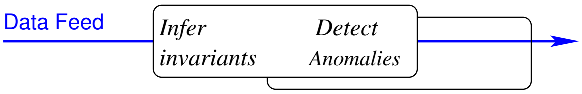

Figure 2: **Anomaly detection usage. Can be fully automated**

Daikon invariants	Mean invariants
(1) `cur` $\geq$ `dlow`	(8) 44.81 $\leq$ `cur` $\leq$ 65.27
(2) `dlow` $>$ `w52low`	(9) 42.05 $\leq$ `dlow` $\leq$ 64.50
(3) `dlow` $\geq$ 48.22	(10) 32.50 $\leq$ `w52low` $\leq$ 32.56
(4) `w52low` $==$ 32.53	(11) 1.43 $\leq$ `beta` $\leq$ 1.43
(5) `beta` $==$ `1.43`	
(6) `cur` $>$ `beta`	
(7) `dlow` $>$ `beta`	

Table 2: **Example of invariants inferred over the data feed of Table 1 by Daikon (left) and by Mean (right)**

a moving window of observations over the stock quote data feed. Table 2 gives an example of invariants Daikon and Mean may infer over the example data feed of Table 1. Each of the inferred invariants is evaluated over observations in the moving window. Table 3 shows an example of detecting an anomaly. The values of `dlow` and of `w52low` are flagged as anomalous because they falsify Table 2 invariants (invariants number (1),(3),(9) and (4),(10), respectively).

User intervention is not required in the usage stage. However, user feedback is desired regarding true positives and false positives, and the severity of true positives. *True positives* are correctly detected anomalous data. *False positives* are normal data falsely detected as anomalous. Such feedback would enable us to further tune the anomaly detector and is useful for repair.

3.2 Repair

In order to repair a system when anomalies are detected, we need to make sure the problem is real and a repair is desired. This often requires user intervention. Only true positives—anomalies that are actually failures—should trigger repair. Further, user feedback can determine repair priority. A repair has high priority for true positives the user cares about. A repair has low priority if the user indicates this is a true positive but not an interesting one (because, for example, it involves attributes the user does not currently use). In addition, user guidance may be needed for repair. For example, a user may need to indicate alternative data feeds to use.

In [24] we discussed sufficient correctness and homeostasis in collections of data feeds, and listed a few alternatives for repair. A repair may result in either restoring and preserving normal operation or operating in a degraded mode. For example, for semantic problems in a data feed, the client may either restore normal operation by using a redundant data feed (a data feed that provides equivalent data) or computing the result another way (e.g., use several other data feeds from which the result can be inferred), or continue to operate in a degraded mode by extrapolating from prior data, accepting degradation of service (e.g., update less frequently), or proceeding without the missing information.

In our example, the user indicates that the anomalies flagged in Table 3 are indeed a failure. There are two possibilities

time	`cur`	`dlow`	`w52low`	`beta`
...	...	...	...	
10/25/2000 11:10	52.38	*8.00*	*8.00*	1.43
...	...	...	...	...

Table 3: **Example of anomaly detection. Evaluating the Table 2 invariants on the data flags the `dlow` and `w52low` values as anomalous**

for handling this failure. The user may currently rely only on the beta value so she indicates this failure is not interesting. As a result, the repair gets low priority and possibly no repair takes place. If, however, the user relies on the values of daily low and 52 weeks low, he indicates this, causing a repair to get high priority. A repair then takes place. For example, if the user provided alternative data feeds, the system uses (possibly temporarily) stock quotes for CSCO from Yahoo!Finance. Otherwise, it extrapolates the current value for each attribute from previous values.

4. TEMPLATES FOR INVARIANTS

Human intervention is essential for adaptation that relies on detection and repair because these require specifications (Section 1). However, a user may not be able to directly provide these specifications, because, for example, it may require too much expertise and time. We propose a template mechanism to elicit knowledge and expectations from users. We expect this mechanism to lower the expertise demands and hope it may also reduce the time demands.

Section 4.1 explains why we believe a template mechanism may be useful. Section 4.2 presents the template mechanism we propose. Section 4.3 gives an example of creating and updating templates. Section 4.4 discusses training and tuning of our augmented anomaly detector.

4.1 Intuition and premises

We assume the user has a purpose in mind for using a data feed and therefore has some expectations for the behavior of the data feed. However, the user may not be able to explicitly state these expectations nor provide useful generalizations about the data. Therefore, we elicit knowledge about the structure of the data feed in the form of skeleton predicates, or *templates*.

The template mechanism can be viewed as a way to elicit good-enough specification proxies from the user. These proxies may serve as a model of normal behavior to support semantic anomaly detection for these data feed and user, or support repair. For anomaly detection, the elicitation takes place mostly in the setup phase. Corrections and refinements are permitted and encouraged during the usage stage. Repair may require eliciting additional information.

Table 6 gives an example of templates that are the result of interacting with the template mechanism. We will describe this interaction shortly in Section 4.3. We found these templates and their classes to be a good-enough specification proxy for the behavior of the example stock quote data feed.

The following premises lead us to believe that templates lower the level of human expertise required. We do not know whether the use of templates reduces the amount of intervention required. Premises related to the nature of human intervention (level of expertise) are: (1) it is easier to understand expectations about the behavior of the data when presented with examples. It is especially useful to examine examples that indicate anomalous behavior, along with the rules for deciding this is anomalous, and (2) it is easier to choose from a list of inferred invariants than to create this list, so having a machine synthesize the list is helpful. Our premise related to the amount of human intervention is that the form of invariants is less likely to change than the parameter values. If this is true, the amount of human intervention required is smaller when using templates. This is surely true for invariants that do not depend on values (e.g., `cur` $\geq$ `dlow`). However, we do not know whether this is the case for adaptive invariants in general.

4.2 Mechanism

The template mechanism asks the user to provide feedback about the *form* of adaptive invariants in a way that indicates whether the user finds a form useful or meaningless. The template mechanism augments the anomaly detection presented in Section 3.1 and the repair presented in Section 3.2 by managing and incorporating user feedback. It does so as follows: (the augmentation is in *italic*)

1. Infer invariants over the moving window observations.
2. Evaluate each invariant over observations in the moving window. Detect an anomaly when an invariant evaluates to false. *Use invariants that*
   - *(a) have been certified (even if they were not inferred for this set of observations) (Item 4d)*
   - *(b) match an existing template (Item 4c)*
   - *(c) are persistent and involve attributes and relations the user did not eliminate (not indicated by Item 4a or Item 4b)*
3. Show to the user invariants together with anomalies detected due to the invariants (including invariants that did not trigger anomalies). It may be helpful to only show a limited number of anomalies per invariant and to rank the invariants by confidence or limit the number of invariants.
4. *Get user feedback. For each invariant, the user may specify one of (and may change prior classifications):*
   - *(a) never use invariants with these attributes and relation (but if you infer a different relation involving these attributes it is OK)*
   - *(b) never use invariants with these attributes together (no matter what relation you think holds)*
   - *(c) (default) set form of the invariant, update values. Different template flavors are possible:*
     - *i. (default) relate attributes only (any relation among these attributes would be considered the same template)*
     - *ii. relate attributes and specify relation (a different relation among these attributes would be considered a different template). This may cause more intervention during usage, but will give finer grain detection*
   - *(d) remember this invariant—I certify this should always hold*

If repair is desired, *a small amount of additional information is needed for the "set form and update values" type of invariants (Item 4c)*. In addition to the user's ability to update the invariant classification during usage, we want to allow the user to *specify an anomaly detected by these invariants is: a false positive (should not have been flagged), a true positive but not interesting, or a true positive and interesting*. Repair is needed only for interesting true positives. It may also be useful to allow the user to specify a finer grain severity than interesting/not interesting, in order to enable better prioritizing of repairs. The repair mechanism may also prompt the user for information specific to a repair strategy.

4.3 Example

The template mechanism maintains a template classification: *accept* for invariants the user accepts as is (Section 4.2, Item 4d), *update* for invariants the user accepts but their values should be updated (Section 4.2, Item 4c), and *reject* for invariants the user rejects (Section 4.2, Items 4b and 4a).

We re-examine one of the stock quote data feed experiments we ran in [23] to show how the template mechanism creates and updates this classification. The templates seem to stabilize rather quickly (around 3 iterations). This indicates the initial template creation can be done during the setup stage.

The Section 2 example is a simplified version of the data feed and invariant inference techniques we used in the experiment. For this data feed, we began with the invariants that Table 2 displays. The template mechanism presents these invariants to the user, along with anomalies each invariant detects (if any). The user then marks each invariant with one of the options presented in Section 4.2, Item 4. Table 4 presents the classification we, as users, chose for each of the Table 2 invariants, along with the resulting templates the template mechanism produces.

Tables 5 and 6 present the results of two additional iterations of template updates. In each iteration, invariants are inferred over new data. The template mechanism asks for user feedback only regarding invariants that have new templates (marked with (*) in the tables). In addition, the user can update existing templates. We explain a subset of our classification decisions. These decisions resulted from gaining a better understanding of the behavior of the data due to evolving invariants.

In Table 5 we get an invariant with the *one of* relation for `w52low`. From the anomalies detected we realize this invariant is better than the one with the '==' relation because `w52low` usually has a few but more than one values in the time period of the moving window. Therefore, we update our previous (Table 4) classification and move the invariant with the '==' relation from the update class to the reject class. Unfortunately, in the next iteration (Table 6) the updated invariant with the *one of* relation includes an anomalous value (8.00). Though it is probably possible to better tune the inference engine, there will always be a trade-off between the detection strength and false positive rate.

Class	Invariant	Template
accept	$\texttt{cur} \geq \texttt{dlow}$	$\texttt{cur} \geq \texttt{dlow}$
	$\texttt{dlow} > \texttt{w52low}$	$\texttt{dlow} > \texttt{w52low}$
reject	$\texttt{cur} > \texttt{beta}$	`cur`, `beta`, any relation
	$\texttt{dlow} > \texttt{beta}$	`dlow`, `beta`, any relation
update	$44.81 \leq \texttt{cur} \leq 65.27$	$\# \leq \texttt{cur} \leq \#$
	$42.05 \leq \texttt{dlow} \leq 64.50$	$\# \leq \texttt{dlow} \leq \#$
	$\texttt{dlow} \geq 48.22$	$\texttt{dlow} \geq \#$
	$32.50 \leq \texttt{w52low} \leq 32.56$	$\# \leq \texttt{w52low} \leq \#$
	$\texttt{w52low} == 32.53$	$\texttt{w52low} == \#$
	$\texttt{beta} == 1.43$	$\texttt{beta} == \#$

Table 4: Our initial classification for the invariants in Table 2. # indicates a numeric valued variable

Class	New(*)/updated invariants	Template
accept		$\texttt{cur} \geq \texttt{dlow}$
		$\texttt{dlow} > \texttt{w52low}$
reject		`cur`, `beta`, any relation
		`dlow`, `beta` any relation
		$\texttt{w52low} == \#$
	(*)$\texttt{w52low} == 1 \bmod 3$	$\texttt{w52low} == \# \bmod \#$
update	$44.99 \leq \texttt{cur} \leq 62.20$	$\# \leq \texttt{cur} \leq \#$
	$29.14 \leq \texttt{dlow} \leq 74.00$	$\# \leq \texttt{dlow} \leq \#$
	$\texttt{dlow} \geq 48.22$	$\texttt{dlow} \geq \#$
	$21.07 \leq \texttt{w52low} \leq 43.30$	$\# \leq \texttt{w52low} \leq \#$
	(*)`w52low` *one of* {32.53,32.56}	`w52low` *one of* {#}
	$\texttt{beta} == 1.43$	$\texttt{beta} == \#$
	$1.28 \leq \texttt{beta} \leq 1.64$	$\# \leq \texttt{beta} \leq \#$

Table 5: Invariant classification after the first update iteration

In the Table 6 iteration we move the `beta` invariant with the '==' relation from the update class to the reject class because this invariant produces too many false positives. We keep the range invariant for `beta` because it better tolerates the small fluctuations which we now believe are normal for the value of `beta`.

In the Tables 4 and 5 iterations we get, for `dlow`, an invariant with the '≥' relation in addition to a range invariant. Though this seems fine at first, the next iteration (Table 6) produces instead an invariant with the '≤' relation. We realize that this invariant reflects the trend of the stock price and this trend changes frequently, whereas the range invariant better describes the behavior of this attribute. Therefore, we move the invariants with the '≤' and '≥' relation to the reject class. As a result of these iterations, the Table 6 templates seem to provide a reasonable description of the behavior of the data.

4.4 Training and tuning

In the anomaly detection setup stage, human intervention is required to help find an effective way to use each candidate technique over the data. This is an iterative process. This process has a number of variables: training set size, attributes to use, and values for tunable technique parameters. We suggest concentrating on finding a good training set size and attribute selection. Once this is done, the setup can be run again with these variables fixed, if the user wishes to find better than default values for tunable parameters (the default is determined by each technique).

In the anomaly detection usage stage, human intervention is desired as feedback regarding true positives and false positives. Repair requires this information. We map the feedback options of the template mechanism (Section 4.2, Item 4) to

Class	New(*)/updated invariants	Template
accept		$\texttt{cur} \geq \texttt{dlow}$
	(*)$\texttt{dlow} \geq \texttt{w52low}$	$\texttt{dlow} \geq \texttt{w52low}$
reject		`cur`, `beta`, any relation
		`dlow`, `beta`, any relation
		$\texttt{w52low} == \#$
		$\texttt{w52low} == \# \bmod \#$
		$\texttt{dlow} \geq \#$
	(*)$\texttt{dlow} \leq 57.25$	$\texttt{dlow} \leq \#$
		$\texttt{beta} == \#$
update	$42.60 \leq \texttt{cur} \leq 63.48$	$\# \leq \texttt{cur} \leq \#$
	$28.35 \leq \texttt{dlow} \leq 73.83$	$\# \leq \texttt{dlow} \leq \#$
	$20.73 \leq \texttt{w52low} \leq 44.78$	$\# \leq \texttt{w52low} \leq \#$
	`w52low` *one of* {32.53,32.56,8.00}	`w52low` *one of* {#}
	$1.32 \leq \texttt{beta} \leq 1.69$	$\# \leq \texttt{beta} \leq \#$

Table 6: Invariant classification after the second update iteration

true positives and false positives. Items 4a and 4b map to false positives, because these are invariants the user eliminates. Item 4d maps to true positives because these are invariants the user certifies should always hold. An attribute value that falsifies such an invariant is therefore an anomaly. Item 4c may produce both true positives and false positives. These are invariants the user finds well-formed yet expects values specified for these invariants to change. We can assume the invariants indicate true positives, unless told otherwise by the user.

5. RELATED WORK

Efforts to increase dependability include prevention and detection/mitigation of problems. Our approach concentrates on detecting semantic problems by the client of a data feed. In general, prevention and detection/mitigation are complementary as complete prevention is rare.

Preventing problems. Our approach deals with the situation as it is today. The Semantic Web [3] suggests a grand vision for a comprehensive solution to syntax/form and semantic failures, requiring many additions to the current Web.

We concentrate on detecting semantic anomalies. Web Services emphasize service discovery and automatic information exchange/integration. This is mainly related to connectivity and syntax/form failures.

Our work can be viewed as concerned with data quality. Most data quality research is concerned with the producer of the data. Our emphasis is on measuring and increasing data quality by the consumer.

Detecting/Mitigating problems. Solutions to all types of problems are needed for improved dependability. Many solutions exist for specific connectivity [12, 4, 11] and syntax/form [15, 6, 16] problems. However, solutions to semantic problems are scarce and either require domain knowledge [22, 19, 6, 16] or provide a specific technique [13].

Our approach of inferring the characteristics of a data feed from its behavior is similar to work in the areas of program analysis [10, 9, 5], testing [7], and intrusion detection [17]. However, these naturally have a different domain, and often concentrate on a specific technique.

Our long term vision is to increase the dependability of software systems through self-healing [25]. The "artificial immune system" [14] concentrates on the security domain. It compares sequences of events as the basic detection method for intrusion detection. IBM's eLiza [2] is an autonomic computing [1] project dealing mainly with connectivity problems in self-managing servers. BEAM [20] is an end-to-end method for real-time fault detection and characterization for deep space probes. It separates the two major forces of sensor data: deterministic and stochastic. Our approach resembles BEAM's approach for the stochastic data.

Making human intervention explicit. Langley [18] recommends that systems supporting computational scientific discovery provide more explicit support for human intervention. He observes that user effort is often required to modulate an algorithm's behavior for the input data. Our template mechanism can be viewed as a way to help users do this. However, our goal is making expectations analyzable, not insights worthy of a scientific publication in their domain.

Lapis [21], a tool for lightweight structured text processing, includes an outlier finder as a way to focus human attention where human judgment is needed. We use anomaly detection in a similar way, though our domain is different.

6. DISCUSSION

Human intervention is inherent in adaptation that relies on detection and repair. We expect the template mechanism to be useful for incorporating human intervention in adaptation and hope this mechanism may also reduce the demands on human expertise and time. We expect this mechanism to help in eliciting expectations from the user and in obtaining information for bridging anomaly detection and repair. The template mechanism could be useful for (1) relation extraction: finding relations that are reasonable among attributes and (2) dimensionality reduction: finding attributes that are meaningful to compare and co-varying attributes. It may also be useful as a means for users to tune the anomaly detection engine for a balance they find comfortable between false positives and true positives.

Open issues related to the template mechanism include verifying the mechanism lowers the demands on human expertise. The expertise level is directly related to our premises presented in Section 4.1. We are exploring necessary refinements and basic implementation details over a specific data feed. A good technical mechanism for incorporating user feedback is necessary, but probably not sufficient, for effectively interacting with a user. The design of a good user interface is challenging, but is beyond the scope of our work.

7. ACKNOWLEDGEMENTS

This research is supported by the National Science Foundation under Grant ITR-0086003, by the Sloan Software Industry Center at Carnegie Mellon University, by the High Dependability Computing Program from NASA Ames cooperative agreement NCC-2-1298, and by DARPA and US Army Research Office under Award No. C-DAAD19 01-1-0646.

8. REFERENCES

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

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

[3] W3C. The Semantic Web, Activity. URL: `http://www.w3.org/2001/sw/`. Accessed November 2001.

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

[5] G. Ammons, R. Bodik, and J. Larus. Mining specifications. In *POPL*, 2002.

[6] M. Bauer and D. Dengler. Trias: Trainable information assistants for cooperative problem solving. In *Intl' Conf' on Autonomous Agents*, 1999.

[7] W. Dickinson, D. Leon, and A. Podgurski. Finding failures by cluster analysis of execution profiles. In *ICSE*, 2001.

[8] Stock quotes data source. URL: `http://quote.pathfinder.com`. Accessed September–November 2000.

[9] 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 *SOSP*, 2001.

[10] M. Ernst, J. Cockrell, W. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. In *IEEE TSE*, 2000.

[11] Google search engine (caching service). URL: `http://www.google.com`. Accessed April 2001.

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

[13] Rulequest. GritBot, autonomous data quality auditor. URL: `http://www.rulequest.com/gritbot-info.html`. Accessed January 2002.

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

[15] C. Knoblock, K. Lerman, S. Minton, and I. Muslea. Accurately and reliably extracting data from the web. In *Data Engineering Bulletin*, 1999.

[16] N. Kushmerick. Regression testing for wrapper maintenance. In *AAAI*, 1999.

[17] T. Lane and C. E. Brodley. Approaches to online learning and concept drift for user identification in computer security. In *KDD*, 1998.

[18] P. Langley. The computational support of scientific discovery. *Human-Computer Studies*, 2000.

[19] K. Lerman and S. Minton. Learning the common structure of data. In *AAAI*, 2000.

[20] R. Mackey, M. James, H. Park, and M. Zak. BEAM: Technology for autonomous self-analysis. In *Aerospace Conference*, 2001.

[21] R. C. Miller and B. A. Myers. Outlier finding: Focusing user attention on possible errors. In *UIST*, 2001.

[22] V. Raman and J. M. Hellerstein. Potters wheel: An interactive data cleaning system. In *VLDB*, 2001.

[23] O. Raz, P. Koopman, and M. Shaw. Semantic anomaly detection in online data sources. In *ICSE*, 2002.

[24] O. Raz and M. Shaw. An approach to preserving sufficient correctness in open resource coalitions. In *IWSSD*, 2000.

[25] M. Shaw. Sufficient correctness and homeostasis in open resource coalitions. In *ISAW*, 2000.