Discovering Software Dependencies in Mobile Ad hoc Networks via Cloud Services

Anitha M[1], Karthik S[2]

[1]Anitha.M, M.E (CSE)/ Ganadipathy Tulsi's Jain Engineering college, Vellore India.

[1]anithasuji15@gmail.com

[2]Karthik S, Asst Prof (CSE)/ Ganadipathy Tulsi's Jain Engineering college, Vellore India.

[2]ramkarthik28@gmail.com

ABSTRACT

Currently the dependencies among the components of service oriented software applications which are running in mobile is difficult to identify due to the frequent mobility of the node and interdependencies. The combination of service-oriented applications, with their run-time service binding and mobile ad hoc networks, with their transient communication topologies brings a new level of complex dynamism to the structure and behavior of software systems. The fixed networks are easy to be monitored in terms of fault analysis and its impact behavior. But identifying the service dependencies on a topology less network under a frequent mobility is difficult to be processed. Hence the project build the new system for dynamic discovery of software dependencies on MANET using cloud services. We evaluate our method in terms of the accuracy of the discovered dependencies and must be capturing snapshots of dependence data to each service request.

Keywords— Mobile ad hoc networks, fault analysis, cloud services.

1. INTRODUCTION

Fault localization and change impact analysis are tasks enabled by accurate and timely data on component dependencies. The importance of dependence information increases with the complexity of the system, both in terms of the number of interacting components required to carry out a given computation and the nature of the environment in which the system operates. In service-based systems, such as those based on the Web Services Architecture, computations are structured as a set of services that respond to requests, where a request typically originates at a user-facing client. The computation fulfilling each request results in a cascade of further requests across some subset of the services. Obtaining dependence information in such a system is made difficult by the inherent loose coupling of services, as many dependencies are unknown at design time, and only established at run time through a dynamic service binding mechanism (so-called "service discovery").The consequence is that the dependencies among run-time instances of services cannot be specified before execution, but instead must be discovered during or after execution. Existing dependence discovery methods focus on statically structured systems operated in fixed networks. A critical assumption made by these methods is that the dependence data, although changing, is relatively stable over time. The significance of the stability assumption is that the methods can make use of statistical techniques based on data collected over long execution periods. Furthermore, by operating in the context of a fixed-network environment, the methods can assume no practical limits on the storage, computational, and communication resources needed to support those statistical techniques. The context for our work is instead service-based systems deployed on mobile ad hoc networks (MANETs). Mobility and ad hoc networking bring increased dynamicity to service dependencies, beyond those caused by the basic service-binding regime. Moreover, the MANET environment is characterized by limitations on the resources available for dependence discovery. Existing methods based on the stability assumption cannot adequately cope with such high levels of dynamicity nor stringent resource constraints. We have formulated a relatively simple method for use in the MANET environment. Our intuition is that dependence discovery should capture snapshots of dependence data relevant to each service request of concern, rather than determine statistical averages for long-term, system-wide dependencies as a whole. Furthermore, the method must be lightweight in its resource usage, which to our thinking means that dependence data should be collected locally, aggregated locally, and drawn to some central location only when and if needed.

We introduce our dependence discovery method and evaluate its sensitivity to a distinguishing aspect of the MANET environment, namely time-dependent behaviour. Clearly, the method is subject to inaccuracies due to such effects as the delay between data collection and data analysis, storage constraints that might require monitors to "forget" some data, and failures in nodes and links. The essence of the present work is to understand how these effects impact the accuracy of dependence discovery, subject to tuning and environmental parameters. Because no benchmarks yet exist for MANET-hosted service-based systems, we develop synthetic data to explore the space of independent variables. The dependent variables in these experiments are the true positives and false positives in the discovered dependence relationships. Mobility and ad hoc networking bring increased dynamicity to service dependencies, beyond those caused by

the basic run-time service-binding regime of SOA or Web Services.

2. OVERVIEW OF EXISTING SYSTEM

Existing dependence discovery methods focus on statically structured systems operated in fixed networks. A critical assumption made by these methods is that the dependence data, although changing, is relatively stable over time. The significance of the stability assumption is that the methods can make use of statistical techniques based on data collected over long execution periods. Furthermore, by operating in the context of a fixed-network environment, the methods can assume no practical limits on the storage, computational, and communication resources needed to support those statistical techniques. The dependence discovery methods can be generally classified as to whether they operate at the network level or at the (application) service level. Network-level discovery focuses on coarse-grained dependencies between network hosts, which are usually described in terms of IP addresses and port numbers. They can be augmented with additional information, such as port mappings or a classification of client applications. On the other hand, service-level discovery focuses on the detection of fine grained relationships between services. Services are hosted in application containers, such as JavaEE and .NET, and typically associated with application identifiers, such as URLs. Understanding the dependencies among the components of a distributed system is critical to making good operational and maintenance decisions. Existing dependence discovery methods focus on statically structured systems operated in fixed networks. A critical assumption made by these methods is that the dependence data, although changing, is relatively stable over time.

3. PROPOSED APPROACH

Proposed system works on the basis of dynamic analysis of dependencies among the web services in MANET's. This helps in identifying and rendering the service only on demand factor. The services are loaded to the cloud pool as a backup. Then based on the location and availability of the nodes, the appropriate service is rendered. This system uses Google App Engine to have service repository. The service is called using RESTful API. The on-demand mechanism is proven to be efficient and accurate. The RESTful Service can be just triggered from the browser console from both web and mobile regardless of native app separately. This solves the existing issue with dynamic discovery of software service dependency.

3.1 API

API describes different web services. Once the request reach the services through HTTP Redirect then API list out the different web services. According to the user request, the service provider provides the services only on demand.

3.1.1 Open Authentication(OAuth 2.0)

In general, this module deals with authenticating the user inside the application, this is considered to be the efficient and secured method in order to allow the user to be authenticated inside the application. An open authentication protocol is an open standard to authorization. It specifies a process for resource owners to authorize third-party access to their server resources without sharing their credentials. Designed specifically to work with Hypertext Transfer Protocol (HTTP), OAuth essentially allows access tokens to be issued to third-party clients by an authorization server, with the approval of the resource owner, or end-user. The client then uses the access token to access the protected resources hosted by the resource server.

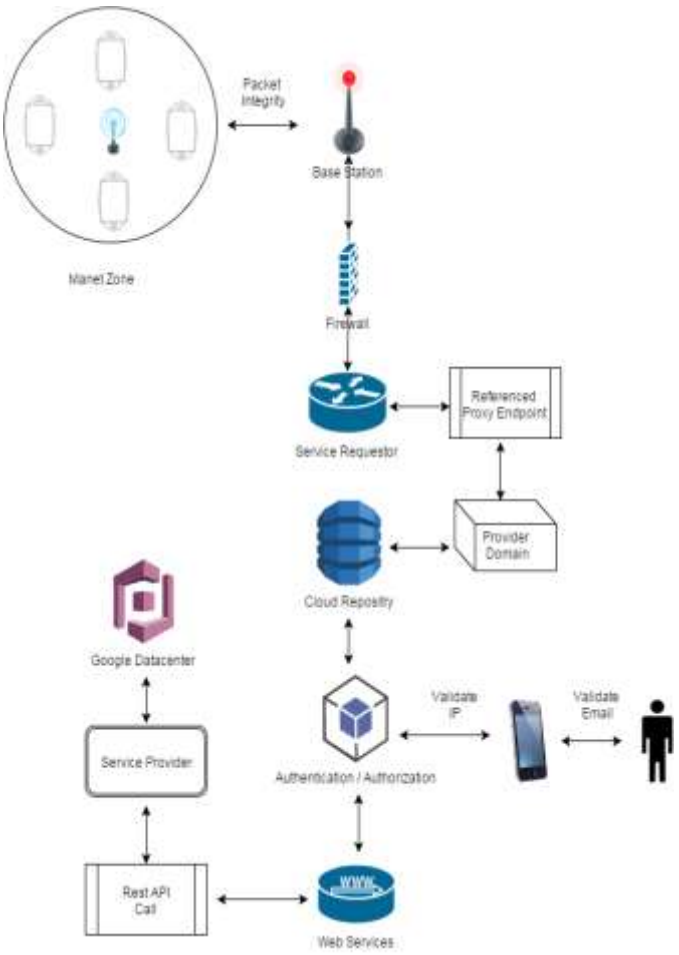

Fig-1: System Architecture

Steps involved in OAuth

1. Obtain OAuth 2.0 credentials

Both Service Provider (SP) like Google and the application know OAuth 2.0 credentials such as a client ID and client secret. The set of values varies based on what type of application you are building. For example, a JavaScript application does not require a secret, but a web server application does.

2. Obtain an access token from the Google Authorization Server

Before your application can access private data using a Google API, it must obtain an access token that grants access to that API. During the access-token request, your application sends one or more values in the scope parameter. There are

several ways to make this request, and they vary by type of application you are building. For example, a JavaScript application might request an access token using a browser redirect to Google, while an application installed on a device that has no browser uses web service requests. Some requests require an authentication step where the user logs in with their Google account. After logging in, the user is asked whether they are willing to grant the permissions that your application is requesting. This process is called user content. If the user grants the permission, the Google Authorization Server sends your application an access token (or an authorization code that your application can use to obtain an access token). If the user does not grant the permission, the server returns an error.

3. Send the access token to an API.

After an application obtains an access token, it sends the token to a Google API in an HTTP authorization header. It is possible to send tokens as URI query-string parameters, but we don't recommend it, because URI parameters can end up in log files that are not completely secure. Also, it is good REST practice to avoid creating unnecessary URI parameter names. Access tokens are valid only for the set of operations and resources described in the scope of the token request.

4. Refresh the access token, if necessary.

Access tokens have limited lifetimes. If your application needs access to a Google API beyond the lifetime of a single access token, it can obtain a refresh token. A refresh token allows your application to obtain new access tokens. Final after getting the token from google, the key is exchanged between the google server and the web application, then if the results matches, then the user is allowed to enter in to the application.

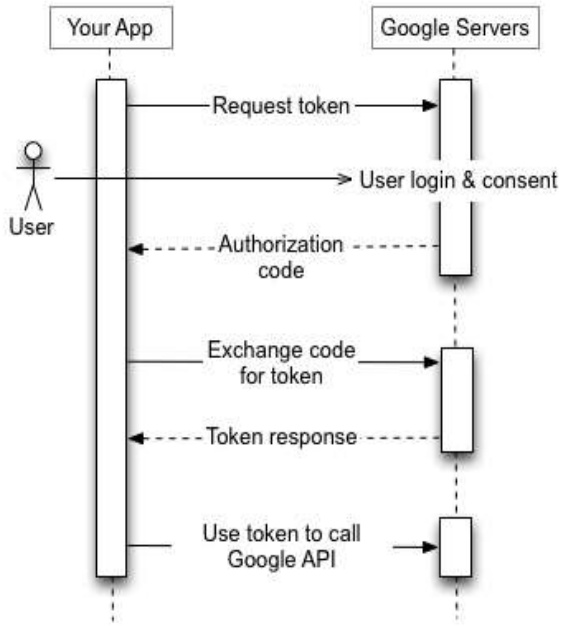

Fig-2: Open Authentication

3.1.2 RESTful API

It possess the following parameters. To the extent that systems conform to the constraints of REST they can be called RESTful. RESTful systems typically, but not always, communicate over Hypertext Transfer Protocol (HTTP) with the same HTTP verbs (GET, POST, PUT, DELETE, etc.) which web browsers use to retrieve web pages and to send the data to remote servers.

1. Client Server

The uniform interface separates clients from servers. This separation of concerns means that clients are not concerned with data storage, which remains internal to each server, so that the portability of client code is improved.

2. Stateless

Statelessness is key which is the necessary state to handle the request is controlled within the request itself as a part of the URI, query string parameters, body or headers.

3. Cacheable

As on the World Wide Web, clients can cache responses. Responses must therefore implicitly or explicitly define themselves as cacheable or not to prevent clients reusing state or inappropriate data in response to further requests.

4. Layered System

A client cannot ordinarily tell whether it is connected directly to the end server or to an intermediary along the way. Intermediary servers may improve system scalability by enabling load-balancing and by providing shared caches. Layers may also enforce security policies.

5. Uniform Interface

The uniform interface constraint defines the interface between clients and servers. It simplifies and decouples the architecture which enables each part to evolve independently.

6. Code on demand

The only optional constraint of REST architecture is code on demand. Servers are able temporarily extend or customize the functionality of a client by transferring logic to it that it can execute. REST really describes the method by which the data is transferred but implementers have been left mostly on their own to figure out how this data should look. It is rapidly changing especially as the number of mobile devices accessing data across the network increases.

3.1.3 Google Cloud Datastore

The Datastore replicates data across multiple datacenters. This provides a high level of availability for reads and writes. Most queries are eventually consistent. The Data store holds data objects known as entities. An entity has one or more properties, named values of one of several supported data types for instance, a property can be a string, an integer, or a reference to another entity. Each entity is identified by its kind, which categorizes the entity for the purpose of queries, and a key that uniquely identifies it within its kind. The Data

store can execute multiple operations in a single transaction. By definition, a transaction cannot succeed unless every one of its operations succeeds; if any of the operations fails, the transaction is automatically rolled back. Cloud data store is a highly-scalable database and it automatically handles replication, providing you with a highly available and durable database that scales automatically to handle your applications load.

1. Simple and Integrated

With cloud datastore RESTful interface, data can easily be accessed by any deployment target. We can build solutions that span across App Engine and Compute Engine, and rely on cloud datastore as the integration point.

2. Fast and Highly Scalable

Focus on building your applications without worrying about provisioning and load anticipation. Cloud datastore scales seamlessly and automatically with your data allowing applications to maintain high performance as they receive more traffic.

3. Easy to Use Query Language

Datastore is a schemaless database, which allows to worry less about making changes to the underlying data structures as the application evolves. Datastore provides a powerful query engine that allows to search for data across multiple properties and sort as needed. Google Cloud Datastore is a fully managed solution for storing non-relational data. Based on the popular Google App Engine High Replication Datastore(HRD), Cloud Datastore provides a schemaless, non-relational datastore with the same accessibility of Google Cloud Storage.

4. MONITORING SERVICES

In order to employ our method, three basic prerequisites must be met. First, to obtain complete dependence information , the monitors should be deployed on the mobile hosts that are either the source or the target of service messages, intermediate hosts in the network used only to store and forward network-level messages are not involved in data collection. Second, the monitors need access to synchronized clocks to allow consistent time-stamping of the collected dependence data. Clock synchronization in MANETs is a well-researched topic, with techniques available to achieve precision of tens or even single microseconds. The shortest period we use for time-stamping data is 6 milliseconds, well within this precision. Third, the monitors must be able to observe service messages and obtain information from those messages, such as client and service identifiers. On the other hand, there is no need for the monitors to have access to the payload of messages. This kind of general information is typically available and visible, since it is used by the underlying service infrastructure to manage service interactions.

Dependencies arise from the flow of messages among services. To discover dependencies, we must therefore track these flows. Because our aim is to be minimally intrusive, we restrict ourselves to observing the message traffic (i.e., messages that contain service requests and responses) as it occurs. Our method makes use of monitors deployed within the network to observe messages and record information about the flows. A convenient place to deploy a monitor is within a service's container. The monitor is then easily aware of the associated service's identity, as well as being provided a context in which to execute.

The main advantage of an approach based on monitors is that it allows us to discover dependencies instantaneously and precisely, with minimal delays between dependence occurrence, detection, and the availability of the dependence information. Moreover, we can do so without having to modify the services themselves. Monitors can also minimize data storage and communication requirements, since they can actively aggregate and summarize the information. Thus, our approach can be thought of as a process for collecting evidence of dependencies, which is in sharp contrast to methods that require storage and transfer of large amounts of data for later statistical analysis.

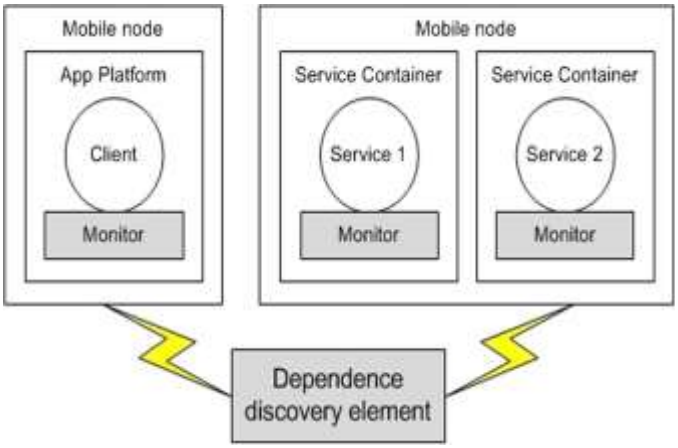

Fig-3: Monitoring Architecture

5. EXPERIMENT

5.1 Simulation of SOA

The simulation of distributed systems that implement a Service-Oriented Architecture(SOA) imposes a set of requirements on their simulation models. These models must replicate the interactions that characterise the SOA operational paradigm and the role of the entities that support an SOA. When those interactions occur over Mobile Ad hoc Networks(MANETs), simulation models must accurately reproduce the effects that mobility and other networking factors have on the system. This paper presents the architecture of a simulation environment for MANET interconnected Service-Oriented Systems. Service call requests issued by consumers to local proxies are forwarded to the distribution container, which formats and transmits service messages. Simulation is the most important and widely used method in the research of Mobile Ad hoc Networks(MANET). The topology of MANET and the mobility of mobile nodes are the key factors that have an impact on the performance of protocols. The network simulation triggers the following three additional events,

Simulation start-up notification: Serves to inform the DES Portal of the availability of the network simulator. This notification is triggered once a warm-up period indicated in the model has elapsed.

Clock tick events: It occurs every millisecond (simulation time). This event is dispatched to the DES Portal with the objective of maintaining the simulation time, mainly for debugging and logging purposes.

Position update events: It collects the positions and speeds of the nodes in ns-2 and pass them to the DES Portal. These are forwarded to the Positioning Service Provider, which notifies those applications in the model that have subscribed to receive location updates.

5.2 Node Deployment

Node deployment can not only reduces the node redundancy and the network costs, but also can prolong the service life of the network. Run-time dynamism is felt even more so when the service-based system is deployed on the mobile nodes of an ad hoc network, where the mobility itself can force reconfigurations of service bindings. Node mobility is one of the most important input parameters that describe the experimental frame in which MANET are evaluated. The simulation environment provides a default implementation of the discovery and distribution APIs for SOA models, built around the specifications.

5.3 Service Deployment

A service deployment is a configuration of a collection of services that specifies the data necessary to instantiate the services as well as dependencies upon other services. The creation of service deployments applications that will deploy or access the services. It contains service deployment groups, service initialization data, service registries and associated named services.

5.4 Service Delivery

Many of the service-level discovery methods apply statistical techniques to traffic traces collected by network hosts and monitors. A dependence is a relation between services defined by the message flow induced by a client request. As an edge case, a dependence is also the relation between a client and a service. Without loss of generality, we mainly focus here on relations among services. The incoming and outgoing inter-dependencies between services is not sufficient to correlate the specific incoming and outgoing inter-dependencies within a service that give rise to intra-dependencies. A monitor is aware of the identity of the source service with which it shares a container, so it can record the outgoing inter-dependence simply by extracting the identifier of the target service from any outgoing request originating at the service and the target service identifier is an essential field present in all requested services.

The dependence discovery element is implemented as a Java library. The element is used by the generic client to discover dependence graphs for conversations initiated by the client. The element uses dynamic web service end-point binding to send messages to the monitor web service hosted on the mobile nodes. The element harvests data from the monitor web services and builds dependence graphs which are stored into a trace file for further analysis.

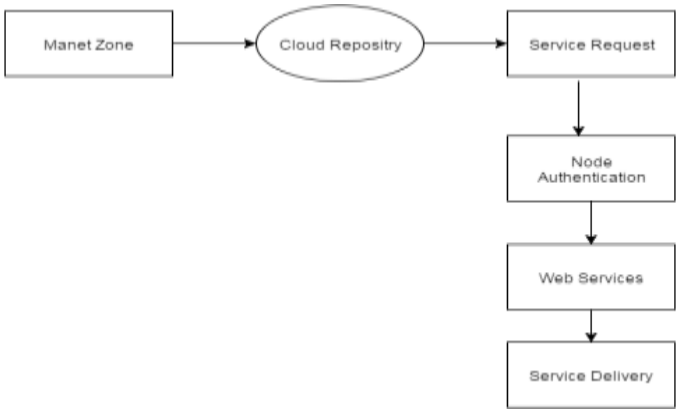

Fig-4: Service Delivery

6. COMPARISON WITH EXISTING METHOD

We now compare the accuracy of our dependence discovery method to that of existing methods. We make this comparison by implementing two alternative methods to represent the two major classes of existing approaches: those that perform discovery at the network level and those at the service level. The service-level alternative discovers a global system dependence graph by observing all the service messages exchanged over the whole execution period and, from this, builds dependence graphs for the individual client conversations. The network level alternative works similarly, but only observes the flow of messages by inspecting the information contained in the headers of packets exchanged over the relevant IP addresses and ports. It then builds dependence graphs using external information provided to it about the deployment of clients and services on hosts.

7. CONCLUSION

This paper presented a run-time method to discover the dependencies among services operated in the highly dynamic and resource-constrained environment of MANETs. Unlike existing approaches, the method does not require stable dependence relationships, nor large amounts of evidence data collected over long periods. Through a set of simulation based experiments, we have evaluated the accuracy of the method in terms of operational factors characteristic of both service-based systems and MANETs. The method exhibits good behavior when subjected to the stress of a changing underlying network topology. Although not reported here, its data storage and data transfer requirements scale well with the number and connectivity of the services involved. Dependence information is not particularly useful in and of itself, but instead serves as a building block for important analysis capabilities. We are currently developing such analyses, including those for probabilistic fault localization and cross-layer performance anomaly diagnosis.

Among the most important characteristics of MANET decentralisation and changing topology can be identified as challenges to the exchange of resources or capabilities. In order to cope with these challenges, the abstraction of shared resources and capabilities as services and the adoption of a Service-Oriented Architecture (SOA) for MANET have been explored as feasible solutions].Additionally, SOA can be considered a building block in the development of mobile cloud applications that may interoperate over a MANET. The simulations are done with different mobility models given the

same geographical layout of network but with different environmental configurations. The scenarios can be visualized on demand in order to study the performance of the network in depth.

REFERENCES

[1] B. S. Baker, "On finding duplication and near-duplication in large software systems," in Proceedings of 2nd Working Conference on Reverse Engineering (WCRE '95), 1995, pp. 86–95.
[2] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees." in Int. Conf. on Software Maintenance, 1998.
[3] K. Kontogiannis, M. Galler, and R. DeMori, "Detecting code similarity using patterns." in Working Notes of 3rd Workshop on AI and Software Engineering, 1995.
[4] J. Krinke, "Identifying similar code with program dependence graphs." in Proceedings of Eighth Working Conference on Reverse Engineering (WCRE '01), 2001, pp. 301–309.
[5] T. Kamiya, S. Kusumoto, and K. Inoue., "Cinder: a multilinguistic token-based code clone detection system for large scale source code." IEEE Transactions on Software Engineering, vol. 28,no. 7, pp. 654–670, 2002.
[6] M. Gabel, L. Jiang, and Z. Su, "Scalable detection of semantic clones," in Proceedings of the 30th International Conference on Software Engineering (ICSE '08), 2008, pp. 321–330.
[7] L. Jiang, Z. Su, and E. Chiu, "Context-based detection of clone related bugs," in Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering, ser. ESECFSE'07, 2007, pp. 55–64.
[8] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, "DECKARD: Scalable and accurate tree-based detection of code clones," in Proceedings of the 29th International Conference on Software Engineering(ICSE '07), 2007, pp. 96–105.
[9] C. Collberg, C. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," The University of Auckland, Tech. Rep.148, Jul. 1997.
[10] C. S. Collberg, C. Thomborson, and D. Low, "Manufacturing cheap, resilient, and stealthy opaque constructs," in Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '98), 1998, pp. 184–196.
[11] C. Wang, "A security architecture for survivability mechanisms,"Ph.D. dissertation, University of Virginia, Charlottesville, VA, USA, 2001, adviser-John Knight.
[12] C. Collberg, G. Myles, and A. Huntwork, "Sandmark–a tool for software protection research," IEEE Security and Privacy, vol. 1,no. 4, pp. 40–49, 2003.
[13] M. Madou, L. Van Put, and K. De Bosschere, "Loco: An interactive code (de)obfuscation tool," in Proceedings of the 2006 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation (PEPM '06), 2006, pp. 140–144.
[14] Semantic Designs, Inc., "Thicket TM," http://www.semanticdesigns.com.
[15] Zelix Pty Lt, "Java obfuscator-Zelix Klass Master," online, http://www.zelix.com/klassmaster/features.html.

