Detecting and Refactoring Operational Smells within the Domain Name System

Marwan Radwan Reiko Heckel

Computer Science Department
University of Leicester
Leicester - United Kingdom

mmmr1@mcs.le.ac.uk reiko@mcs.le.ac.uk

The Domain Name System (DNS) is one of the most important components of the Internet infrastructure. DNS relies on a delegation-based architecture, where resolution of names to their IP addresses requires resolving the names of the servers responsible for those names. The recursive structures of the inter-dependencies that exist between name servers associated with each zone are called dependency graphs. System administrators' operational decisions have far reaching effects on the DNSs qualities. They need to be soundly made to create a balance between the availability, security and resilience of the system. We utilize dependency graphs to identify, detect and catalogue *operational bad smells*. Our method deals with smells on a high-level of abstraction using a consistent taxonomy and reusable vocabulary, defined by a *DNS Operational Model*. The method will be used to build a diagnostic advisory tool that will detect configuration changes that might decrease the robustness or security posture of domain names before they become into production.

1 Introduction

The Domain Name System (DNS) [20] is critical to the integrity of Internet services and applications. The DNS is a distributed database for storing information on domain names, the primary namespace for hosts on the Internet. The name space is organised in a hierarchical structure to ensure domain name uniqueness. Each node in the DNS tree corresponds to a zone. Each zone belonging to a single administrative authority is served by multiple authoritative name servers.

The correct and error-free operation of the DNS is crucial for the reliability of most applications on the Internet. Operational guidelines [4, 11, 1] require that a zone have multiple authoritative name servers, and that they be distributed through diverse network topological and geographical locations to increase the reliability of that zone as well as improve overall network performance and access. It also makes DNS services robust against unexpected failures. Recent work [10, 23] outlines the need for zone operators to understand how many inter-dependencies they may inadvertently be incurring through the deployment and sharing of DNS secondary servers.

The original DNS design focused mainly on system robustness against physical failures, and neglected the impact of operational errors such as misconfigurations and bad deployment choices. Several previous measurements [24, 14, 26] showed that zones with configuration errors suffer from reduced availability and increased query delays up to an order of magnitude. DNS administrators have to decide on operational parameters and be aware of their implications for the DNS's overall system qualities. On the deployment level, configuring the number of redundant authoritative DNS servers for a certain zone shall take into consideration the operational overhead associated with querying multiple servers in parallel. Choosing servers with names under other zones provides zone redundancy but may incur security and resiliency threats to the zone. Deciding on where to physically locate the servers should ensure a

Preliminary Report. Final version to appear in:
1st Graphs as Models (GAM) Workshop, ETAPS-2015

certain degree of resistance against different types of failures. Peering with external organizations for secondary server hosting should take into consideration the impact of transitional trust and administrative complexity [25, 13].

While the original DNS design documents [16, 20, 21, 4, 11] call for diverse placement of authoritative name servers for a zone, bad configurations may lead to *cyclic dependencies* while bad deployment choices may lead to *diminished and false server redundancy*. It was also assumed that redundant DNS servers fail independently; previous measurements [25, 10] showed that operational deployment choices made at individual zones can introduce *excessive zone influence* that severely affect the availability, security and resiliency of other zones.

This research is motivated by the lack of formal analysis of the DNS interdependencies stemming from the delegation-based architecture as well as operational deployment choices made by system administrators. We approached the problem from a design point of view that takes into consideration the DNS zone configuration and server deployment choices rather than from the dynamic behavioural view [7] which includes statistical and post-deployment measurements. We propose a method to identify, specify and detect misconfigurations and bad deployment choices in the form of operational bad smells.

The method utilizes a set of structural metrics defined over a DNS operational model to detect the smells in early stages of the DNS deployment. It also suggests graph-based refactoring rules as correction mechanisms for the bad smells. We apply and validate the method using several representative case studies. The method will be used to build a *pre-emptive diagnostic advisory* tool that will detect and flag configuration changes that might decrease the robustness or security posture of a domain name, before even the changes become into production.

The contributions of this research are:

1. Introduction of the concept of operational bad smells, i.e., recurring DNS deployment bad choices and misconfigurations that have negative impact on certain aspects of the DNS's quality.

2. Description in detail of a set of representative operational bad smells to build a DNS operational bad smells catalogue.

3. Identification of a set of structural metrics, defined over a DNS operational model, to query the dependency graph of the system to detect DNS operational bad smells.

4. Suggestion of graph-based refactoring rules as correction mechanisms to eliminate the bad smells.

The rest of the paper is structured as follows: Section 2 discusses relevant background. Section 3 presents the DNS operational model. Section 4 discusses the bad smells' identification, specification, detection and refactoring method. Section 5 validates our method by applying it to a set of representative case studies. Section 6 discusses some related work and Section 7 concludes the paper and discusses future work.

2 The Operation and Structure of the DNS

DNS is responsible for the mapping of human-friendly domain names to the corresponding machine-oriented IP addresses. Operators of each zone determine the number of authoritative name servers and their placement and manage all changes to the zone's data content. In spite of the fact that zone administration is autonomous, some coordination is required to maintain the consistency of the DNS hierarchy.

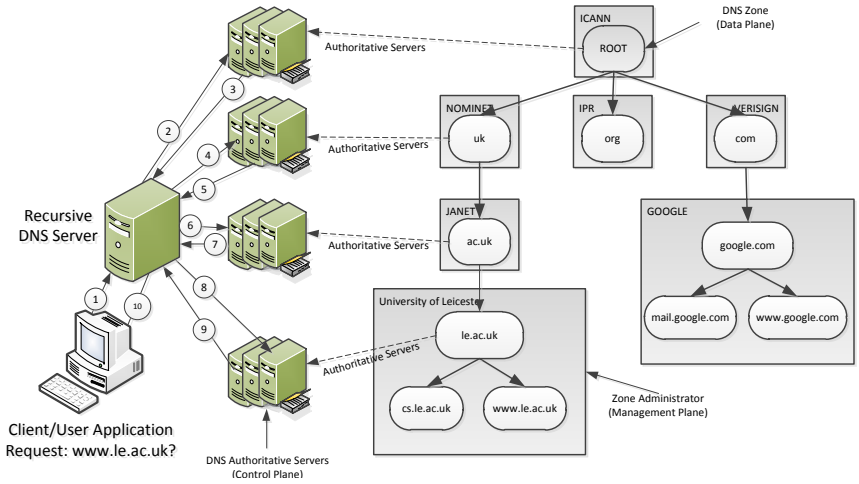

Figure 1: An illustration of the DNS resolution process.

2.1 General Operation of the DNS

Figure 1 shows the process by which an application looks up the domain name www.le.ac.uk and how it is mapped to the DNS data, control and management operational planes. To find the IP address of www.le.ac.uk, the client (e.g a web browser) submits a DNS query to a recursive DNS resolver (step 1). Assuming that the corresponding IP is not in the resolver cache, it will ask one of the root name servers for the translation (step 2). The names and IP addresses of root name servers are locally stored within each server. The root name servers will respond with a referral, telling the resolver to query the DNS servers of the .uk domain for an answer (step 3). The resolver then repeats this process for the .uk name servers and get a referral to ask the .ac.uk name servers which in turn answers with a referral to as the le.ac.uk name servers (step 4 -7). The resolver next asks one of the le.ac.uk name servers for the translation (step 8), and gets the answer in step (9), and finally forwards the answer to the requesting client (step 10) who will use this information to connect to the web server hosting the web site www.le.ac.uk. Throughout the process, resolvers may encounter name servers hosted under other zones whose names need to be resolved before contacting them about the original request.

2.2 DNS Operational Inter-dependencies

Inter-dependencies are common in the DNS and stem from the hierarchal structure of the DNS, the DNS protocol as well as from different motivations and goals [10]. A zone is said to depend on a name server if the name server could be involved in the resolution of names in that zone. The dependencies among name servers that directly or indirectly affect a zone are represented as a dependency graph.

Figure 2 shows the delegation graph of the zone (le.ac.uk) where the zone le.ac.uk depends on 4 authoritative name servers (ns0, ns1 and ns2.le.ac.uk) under the management of the University of Leicester (UoL), while the fourth name server (adns0.bath.ac.uk) is managed by the University of Bath. In order to resolve any domain under the zone (le.ac.uk), resolver will ask the name servers of the root zone down to the set of authoritative name servers of the zone. While Leicester University directly trusts bath.ac.uk to serve its namespace, it has no control over the name servers that Bath trusts (i.e. name servers under Cambridge, Salford, and Imperial College and so on). Each name server or group of name servers are administered by different organization which creates another layer of transitive trust dependencies amongst those organizations.

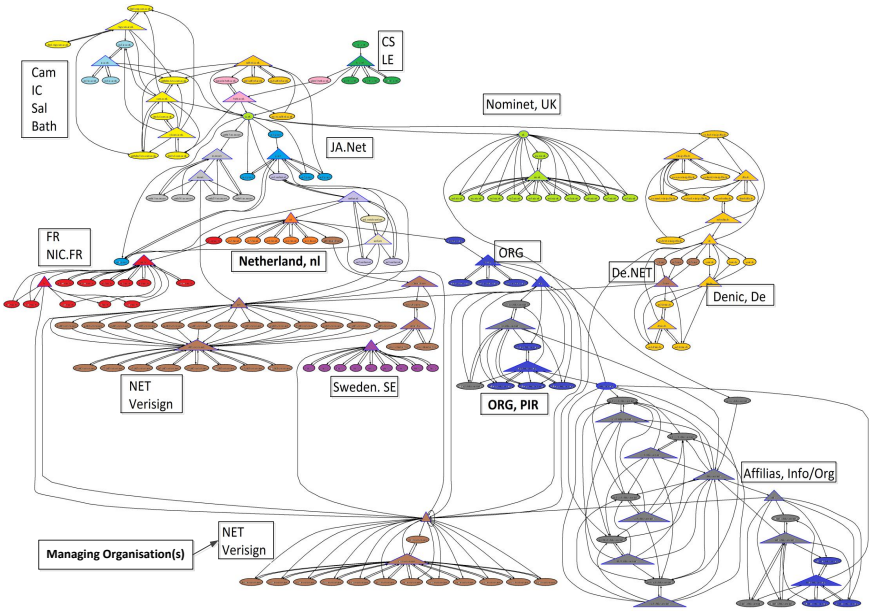

Figure 2: Name Dependency Graph of (le.ac.uk).

2.3 Operational Planes

The zone's data plane is the interconnected graph of all infrastructure resource records defined within the zone's configuration file. The interconnected graph of all authoritative name servers involved in the resolution process of a domain within a certain zone is called the zone's control plane and the interconnected graph of all administrative units involved is called the management plane. One reason that the DNS is so powerful is that its data plane allows administrators a great deal of flexibility: they can manage their name space however they like. However, the control and management planes' flexibility can lead to operational problems if not managed conscientiously.

2.4 Dependency Graphs

The recursive structures of inter-dependencies within and between the DNS operational planes is represented by dependency graph. A dependency graph [10] is a directed connected graph with a distinguished node (r) which is the root zone. Each node in the graph represents a zone name, and each edge signifies that its source is directly dependent on its target for proper resolution of itself and any descendant domain names. Dependency graphs capture most attributes and relationships between the various operational entities within the DNS and they can be effectively utilized in detecting configuration weaknesses and servers deployment problems. Figure 3 shows deferent dependencies that occur at the different DNS operational planes.

Since many of the misconfigurations can't be detected from the zone file or deployments directly, there is a need for an operational model that encompasses all information related to the zone file and the server deployments in one conceptual graph. The instance of the model (the dependency graph) will enable us to detect zone integrity violations as well as violations in the deployment of name servers and the choice of peering organizations and management structures. The conceptual graph representation facilitates modelling at multiple levels of details simultaneously.

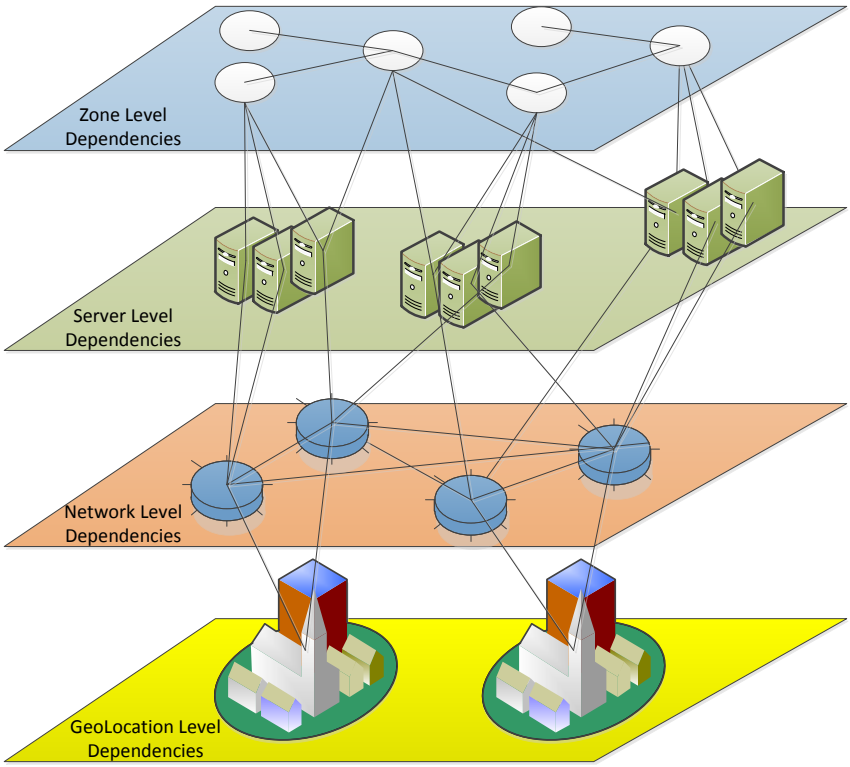

Figure 3: DNS Operational Planes and Their Dependencies.

3 DNS Operational Model

The DNS Operational Model aims to support operational goals, such as detecting violations of the design and deployment principles, at the authoritative level. To this end we have to search for certain patterns indicating such violations in the instances of the operational model of the system, i.e., the dependency graphs. This means we have to be able to specify a problem as a pattern, and to query the dependency graph about the existence and occurrences of the specified pattern.
The model is composed of the following elements:

- Operational Entities (e.g. resource records, zones, servers and organizations)
- Properties of operational entities such as (in-bailiwick and out-of-bailiwick name servers)
- Relations between the entities (e.g. access attributes such as dependability, containment, delegation and management)

The operational DNS entities that appear in our model fall into two categories: primitive and composed entities. Composed entities have an identity and a set of properties. In addition to these, composed entities have a list of contained entities, which are primitive or composed entities. A composed entity type is one that contains other entities. The model supports the following composed entities: Organization, Server, Zone and Resource Record. In order to describe a composed entity we have to specify its properties, containment structure (i.e. the entities that it contains), relations and container entity. As an example, we can look at the server component where it can be managed (contained) by organizations. Multiple servers can be managed by one organization. The server can host many zone files and it has

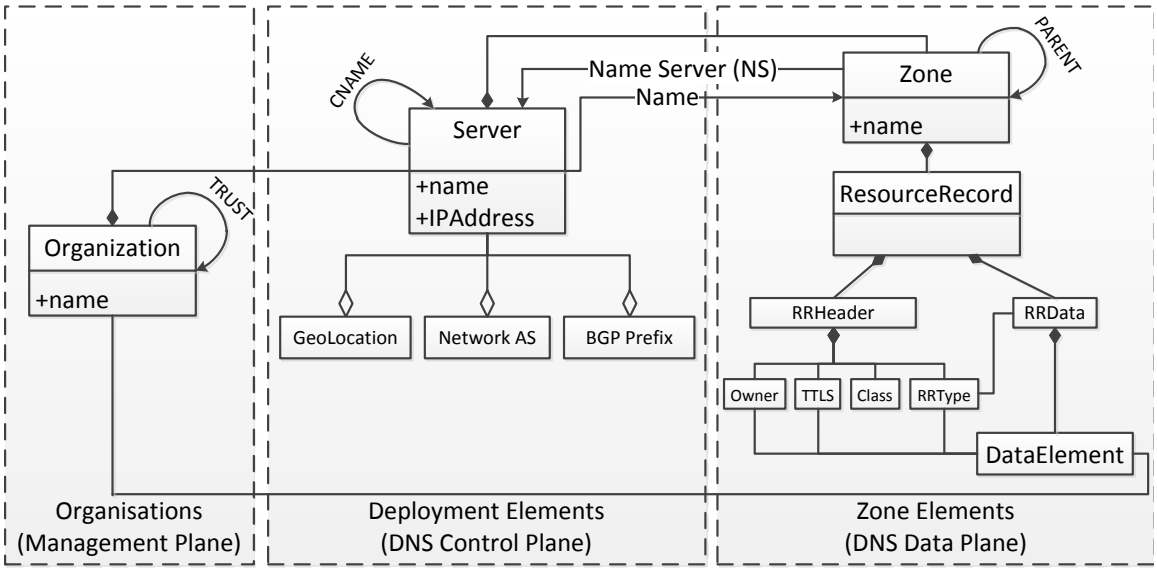

Figure 4: DNS Operational Model.

the name and IP address as attributes. There are many types of servers and in this context we are concerned with in-bailiwick servers whose name is within the zone file hosted at that particular server and out-bailiwick servers who has a name from a zone hosted in another name server.

Three specific dependencies are present within the DNS operational planes and they are:

1. Parent Dependency: resolving the name of a domain name is always dependent on resolving its parent name since the resolver must learn the authoritative servers for a zone from referrals from the zones hierarchical parent.
2. Authoritative Name Server (NS) Dependency: A zone is said to depend on a name server if the name server could be involved in the resolution of names in that zone.
3. CNAME Aliasing Dependency (Name pointing to another Name): the resolution of an alias is always dependent on the resolution of its target CNAME. If a resolver receives a response indicating that the name in question is an alias to another name, it must subsequently resolve the target of the alias, and so on until an address is returned.

The dependency graph can be extracted from the zone file and from the chain of authoritative name servers and organizations involved in the resolving process of domains under that particular zone. This is done by analysing the zone file and the dependencies between the different resource records and their data elements and by following the query process as outlined in Fig. 1 using certain DNS tracing tools extensively. All types of dependencies and recursive queries are followed to get the full dependency graph of the zone in the three operational planes.

4 Operational Bad Smells

In software engineering, bad smells in code [12] identify risks to non-functional quality in a software system based on structural properties and metrics. We transfer these ideas to the realm of the DNS, where operational bad smells are configuration and deployment choices by zone administrators that are not errant or technically incorrect, and do not currently prevent the system from doing its designated

functionality. Instead, they indicate weaknesses that may impose additional overhead on DNS queries, or increase the system vulnerability to threats, or increase the risk of failures in the future.

The set of identified bad smells is being formally specified in concise and reusable terms based on a template that includes the bad smell name, type, inspection plane(s), description & occurrences, quality impacts and detection strategies. The catalogue will be expanded by including refactoring rules for each smell and how these rules have to be applied on the model instance to eliminate the concerned bad smell. Examples of catalogue entries are shown in Table 3 and Table 5 listed as part of the case studies in Section 5.

Although DNS troubleshooting techniques and problem identification methods have been proposed and several tools have been built, most of these methods and tools apply their detection techniques directly on the zone files through a predefined zone schema and integrity constraints. They don't take into account the inter-dependencies stemming from the hierarchical nature of the DNS or the zone administrators practices. Instead, we propose a model-based approach that subsumes all the steps necessary to identify, specify and detect the DNS operational bad smells. The ***ISDR*** method is composed of four stages and produces the operational bad smells catalogue:

1. **I**dentification, including domain analysis using DNS standards in the form of Request for Comments (RFCs), best practices and policy documents, literature review and DNS expert views.
2. **S**pecification of a set of operational bad smells using a reusable vocabulary and classification of the bad smells in a taxonomy that shows the scope of the inspection element or plane and system's external qualities affected by the smell.
3. **D**etection of bad smells in the form of general detection queries and formulas.
4. **R**efactoring as a correction mechanism to the operational bad smells. Other correction mechanisms may be formulated in the form of reports or reconfiguration recommendations.

The following are the ISDR method stages in details:

4.1 Identification

The first stage in our method consists of performing deep analysis of the DNS standards, Request for Comments (RFCs), best practices and policy documents to identify weaknesses in configuration and deployment choices made by administrators that may impose additional overhead on DNS queries, or increase the system vulnerability to threats, or increase the risk of cascaded failures.

4.2 Specification

The weaknesses identified in the previous step, termed as *operational bad smells*, are then defined using certain key terms, unified vocabulary and reusable concepts in this domain. We developed a taxonomy that describes the structural relationships between the various bad smells. The taxonomy has an important role in defining the scope of inspection and highlighting the metrics or structural properties related to the bad smell. It classifies the bad smells based on the following categories:

1. Operational plane: Data, control and management planes.
2. Affected entity types: Single type, inter-type, intra-type, or inter-zone.
3. Property of the smell: Lexical, structural or measurable.

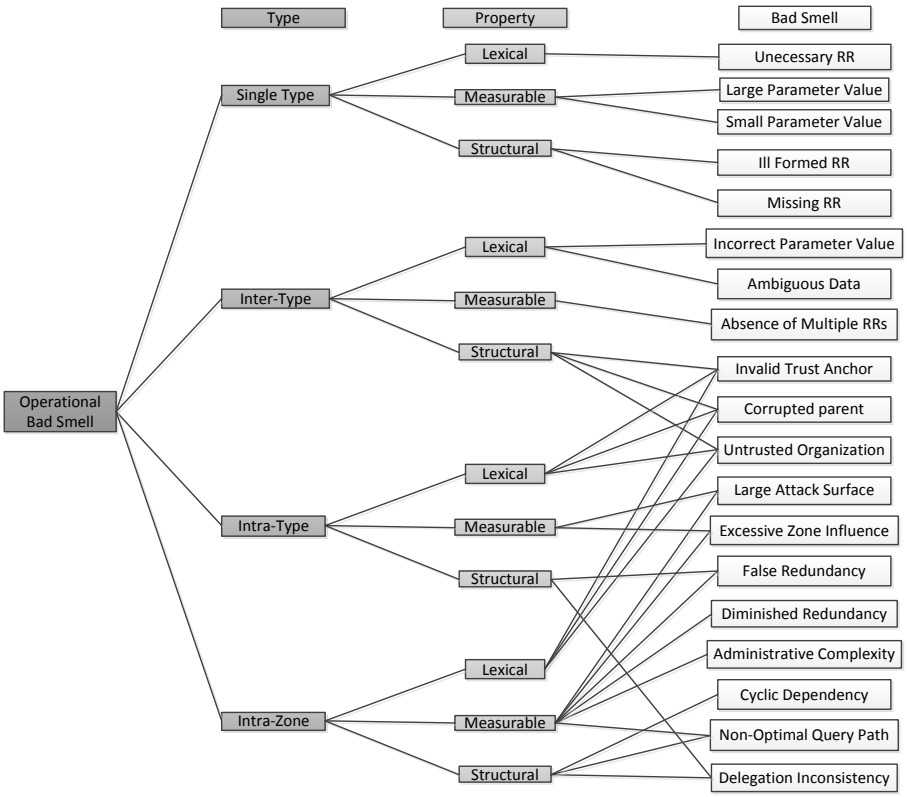

Figure 5: DNS Operational Bad Smells Taxonomy.

Figure 5 shows a partial graphical representation of the DNS operational bad smells taxonomy. The taxonomy is generic and defines a bad smell in more than one category. It can easily be extended by defining new categories of bad smells based on subsequent iterations of the DNS operational domain analysis. So far we have already identified 19 bad smells that can be used as a representative set that spans the different operational planes with various detection properties.

In the context of metrics-based analysis techniques, the aforementioned classification of design entities (as explained in Section 3) has a particular relevance: it provides a pertinent explanation about why metrics are defined and computed only for some entity types (i.e., organizations, network, server, zones and resource records). The explanation resides basically in the distinctive aspects that exist between the two, i.e., the fact that a composed entity can contain other entities and that it can have relations with other entities. As direct measurements are mainly counting the different entities contained in, or related to a measured entity, it becomes obvious why the object of measurement is restricted to composed design entities. Interesting examples of metrics are per-server and per-zone distributions such as:

1. The number of zones that are served from multiple name servers in different network autonomous systems or diverse geographical locations *(Server Redundancy)*.
2. The number of zones that influence the resolving of domain names within a particular zone *(Zone Influence)*.

For the proper interpretation of each structural metric defined over the operational model, we give the metric definition, usability, how to measure and a formula for computing that metric. Table 1 shows the interpretation model for the metric *Administrative Complexity* [10].

Table 1: Interpretation of the Administrative Complexity Metric.

Metric	Administrative Complexity.
Definition	Describes the diversity of a zone with respect to the organisations administering its authoritative name servers.
Usability	Mutual hosting of zones between organizations is common in the DNS deployment schemes. The advantage of such practice is an increased availability but at the same time increased potential of failure and instability of the DNS zone resolution process.
How to Measure	Count the number of authoritative name servers of each organization involved in the dependency graph of zone(z).
Metric Notation	O_z denotes the set of organizations administering authoritative name servers hosting the zone (z); n denotes the total number of authoritative name servers of zone (z); NS_z^o denotes the subset of name servers administered by organization o in O_z.
Formula	$Ac(z) = \sum_o^n (\frac{NS_z^o}{NS_z})^n$.

4.3 Detection

In order to be able to detect bad smells occurring on model instances, we need to capture deviations of the particular instance model from the good and recommended operational best practices. Lexical and structural properties are used to detect some of the bad smells using direct queries on the instance model such as (Are there any cycles in the dependency graph.?).The metric-based approach combines a set of metrics and set operators to compare them against absolute or relative threshold values. Setting the absolute or relative operational metrics threshold values can be done using local policy constraints or best practices from the wider DNS domain literature and expert views.

4.4 Refactoring

In the area of object-oriented programming, refactoring [22] is the technique of choice for improving the structure of existing code without changing its external behaviour. Graph-based, general refactoring rules [6] will be suggested to remove the bad smells identified and detected in the previous stages. The general approach of refactoring [19] is to include the following steps: (1) identify the location for refactoring, (2) determine which refactoring rules should be applied and on what sequence, (3) guarantee that refactoring rules are preserving the external behaviour of the system, (4) application of selected refactoring rules, (5) assess the effect of refactoring on the systems external qualities and (6) maintain the consistency between the refactored elements and other system artefacts.

4.5 Method Execution and Tool Support

The ISDR method is executed on a particular instance of the DNS operational model (Dependency Graph) using the following steps:

- **Step 1:** Extract the dependency graph from the zone configuration file and the authoritative servers' deployment.
- **Step 2:** Query the dependency graph for any bad smell using the methods and metrics defined in the Bad Smells Catalogue.

- **Step 3:** Apply relevant refactoring rule(s) on all matching occurrences of the LHS of the rule on the instance model. A new dependency graph is generated as an output of this step.
- **Step 4:** New zone file(s) and authoritative name servers deployment layout can be automatically generated from the new Dependency Graph or a set of recommendations can be presented to the system administrator with relevant quality impacts.

The method will be implemented using a pre-emptive diagnostic advisory tool that will detect and flag configuration changes that might decrease the robustness, resilience or security posture of a domain name, before even the changes become into production.

5 Validation

We validate our method by applying it and its associated execution technique to several bad smells where some of them have been already identified as misconfigurations in the literature [24, 10, 14, 13, 17].

5.1 Case Study (1): Cyclic Dependency

To achieve acceptable geographical and network diversity, zone administrators often establish mutual arrangement with peer organizations to host each others zone files. Authoritative name servers located in other zones are normally identified by their names instead of their addresses and called out-of-bailiwick name servers. A cyclic zone dependency [24] occurs when two or more zones depend on each other in a circular way.

Table 2 shows that the zone (example.com) has 4 authoritative name servers responsible for resolving domain names under this zone as defined in its parent zone (.com). Two servers (ns1 and ns2.example.com) are *in-bailiwick* servers and it is mandatory to include their IP addresses in the parent zone in order to properly resolve domain names under that zone. The other two servers (dns1 and dns2.example.net) are located in another zone and there is no need to include their IP addresses in the (.com), example.com parent zone file. On the other hand, the (.net) zone which is the parent of the (example.net) zone, is served by two *out-of-bailiwick* name servers located in the (example.com) zone.

Table 2: Content of Zone File for Case Study (1).

$ORIGIN .com.		
example.com.	NS	ns1.example.com.
example.com.	NS	ns2.example.com.
example.com.	NS	dns1.example.net.
example.com.	NS	dns2.example.net.
ns1.example.com.	A	1.1.1.1
ns2.example.com.	A	1.1.1.2

$ORIGIN .net.		
example.net.	NS	ns1.example.com.
example.net.	NS	ns2.example.com.

In this example, the two zones work nicely under normal circumstances but if (for any reason), both in-bailiwick name servers become unavailable, both example.com and example.net zones will not be reachable because the IP Addresses of the other two authoritative name servers can't be resolved. This example illustrates the failure dependency between zones, where the failure of some servers in one zone will render the other zone unreachable. The quality impacts of such a bad smell are significant reduction on availability and resiliency of the zone against multiple points of failure.

Checking each zone individually for configuration errors will not lead to the detection of this bad smell since they are both configured correctly. On the other hand, constructing the dependency graph will easily show the occurrence of two circular paths that identify the smell.

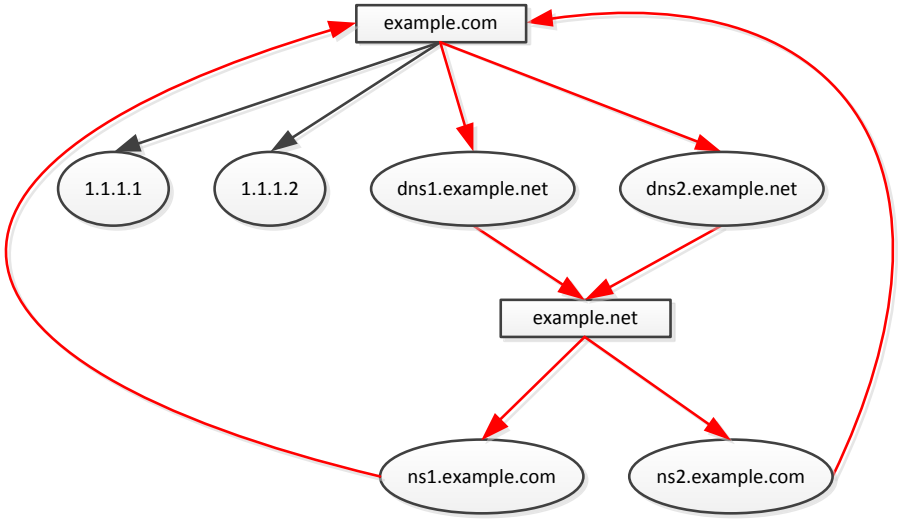

Figure 6: Part of the Dependency Graph of Case Study (1).

Table 3: Catalogue Entry for the Cyclic Dependency Bad Smell.

Name	Cyclic Dependency.
Type	Intra-Zone, Structural.
Inspection Planes	Data and Control Planes.
Occurrences	Cyclic zone dependency occurs when two or more zones depend on each other in a circular way.
Quality Impacts	Reduced availability and reduced resiliency.
Detection Strategy	Is there any cycle in the Dependency Graph? (Query on the DNS Operational Model Instance).
Correction Mechanism (Refactoring)	Add a glue record for the (out-of-bailiwick) authoritative name servers involved in the cycle in the zone file.

Cyclic Dependencies can be eliminated by the inclusion of specific resource records (RRType: A) for both out-of-bailiwick servers in the (.com) zone. This will enable resolving the domain names under the (example.com) and (example.net) zones even when the two in-bailiwick servers are unreachable. We execute this correction mechanism in the form of a graph transformation based refactoring rule (AddGlueRecord) applied on the dependency graph as shown in Figure 7. Since we have two matches for the LHS of the rule on the actual instantiation of the model (the dependency graph in Figure 6), then the rule needs to be applied twice in order to remedy all occurrences of the bad smell. A new zone file can then be automatically generated from the new Dependency Graph as shown in Table 4 or a set of recommendations can be presented to the system administrator to eliminate the bad smell.

5.2 Case Study (2): False Redundancy

Redundancy [11] is one of two mechanisms used by DNS administrators to ensure high availability of domain names. The level of availability provided by redundant servers is a function not only of their number, but also of their physical location and the networks they connect to. In 2001, a DNS bad deployment choice [9] caused many Microsoft's web sites and email servers to be unreachable (although they were actually still operational). All authoritative name servers for the zone (microsoft.com) were

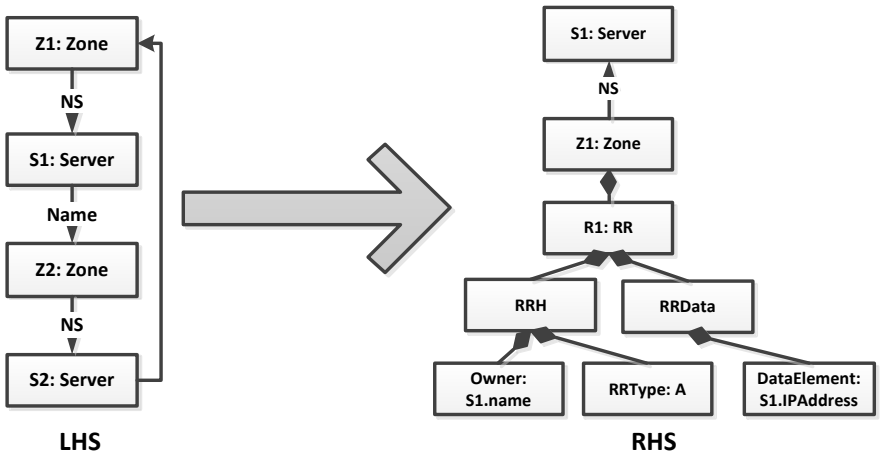

Figure 7: Refactoring Rule: AddGlueRecord.

Table 4: New Zone File Generated After Executing the Refactoring Rule(s).

$ORIGIN .com.		
example.com.	NS	ns1.example.com.
example.com.	NS	ns2.example.com.
example.com.	NS	dns1.example.net.
example.com.	NS	dns2.example.net.
ns1.example.com.	A	1.1.1.1
ns2.example.com.	A	1.1.1.2
dns1.example.net.	A	1.1.1.3
dns2.example.net.	A	1.1.1.4

$ORIGIN .net.		
example.net.	NS	ns1.example.com.
example.net.	NS	ns2.example.com.

place in one location, connected to the same network, and placed behind one particular network router. When the router failed, this local bad choice has a large global impact by increasing the queries on one of the DNS root servers (F server) from the normal 0.003% of all queries to over 25% [24]. The catalogue entry for the False Redundancy bad smell is shown in Table 5.

In this example, we are looking into one aspect of False Redundancy, which is the geographical placement of the authoritative name servers. Looking at the dependency graph extracted from the zone file, generated as an output of case study (1) and the deployment of authoritative name servers, as shown in Figure 8, it is clear that the *geographical redundancy* of the zone (example.com) is one which is much less than the server's Redundancy which is supposed to be 4 (the total number of authoritative name servers defined in the zone). Looking at the IP address associated with each of these servers, it is evident that all of them are connected to the same network, and behind even the same router. This deployment choice introduces a single point of failure since all servers are located in the same geographical area and this badly affect the resiliency and availability of the zone and its domain names. Geographical area may be defined as a continent, country, city or even a certain building, which may also be susceptible to power outage, natural disaster or any other risk.

In order to detect the occurrence of the False Redundancy bad smell, one set of queries regarding the number of authoritative name servers of the particular zone, number of distinct geographical locations

Table 5: Catalogue Entry for the False Redundancy Bad Smell.

Name	False-Redundancy.
Type	Measurable and Inter-zone.
Inspection Planes	Control Plane.
Description	In order to gain full control over the management of authoritative name servers, a system administrator may decide to put all authoritative name servers within one location and under the same network. The level of availability provided by redundant servers is a function not only of their number but also of their physical location and the networks they connect to.
Occurrences	When all redundant servers are located within the same physical location, connected to the same network, placed within the same address prefix.
Quality Impacts	Reduced availability, decreased resilience, and the system become susceptible to single point of failure at certain granularity.
Detection	Queries on the dependency graph regarding the following metrics: a) number of authoritative name servers, b) number of geographical locations servers are placed in, c) number of networks those servers are connected to, and d) distinct BGP prefixes,
Refactoring	Applying the MoveServerLocation refactoring rule will ensure the availability of the zone and its resilience to a single point of failure.

where those servers are placed in, can be executed against the dependency graph in Figure 8. The resulted measurements are used in detecting the bad smell as defined in its catalogue entry. The threshold values for the metrics are set based on the best practices and policies as identified in the first step of the ISDR method or can be left to the system administrator to set based on the local policies and operational requirements.

It should be noted that the refactoring rule shown in Figure 9 is just one of the options to eliminate this bad smell, i.e., there could be another rule for creating a new server in a new location rather than moving an existing one. We can look at network number or BGP prefix instead of location. It can also take more than one rule application to resolve the situation, so a single rule specifies an incremental improvement, which may have to be repeated or combined with others.

6 Related Work

Ramasubramanian, et al. [25] demonstrate the far-reaching effects of DNS dependencies. Their results show that a domain name relies on 44 name servers on average. Deccio et al. [10] perform further examination of name resolution behaviors to create a formal model of name dependencies in the DNS and quantify the significance of such dependencies. Several surveys of production DNS deployments have been conducted [24, 14, 26] and various misconfigurations are analyzed. So far the main efforts in addressing the problem have focused on informing the operators about the existence of DNS configuration errors, either by Internet RFCs [4, 11] or with directives set by specific organizations [1].

Chandramouli and Rose [8] considered integrity constraints pertaining Resource Records (RRs) from single and multiple zones. They found that many integrity constraints have to be satisfied across zones. Casalicchio, et al. [7] proposed a set of metrics to be used to evaluate the health of the DNS system by measuring the DNS along three dimensions, namely vulnerability, security and resiliency. Despite all the

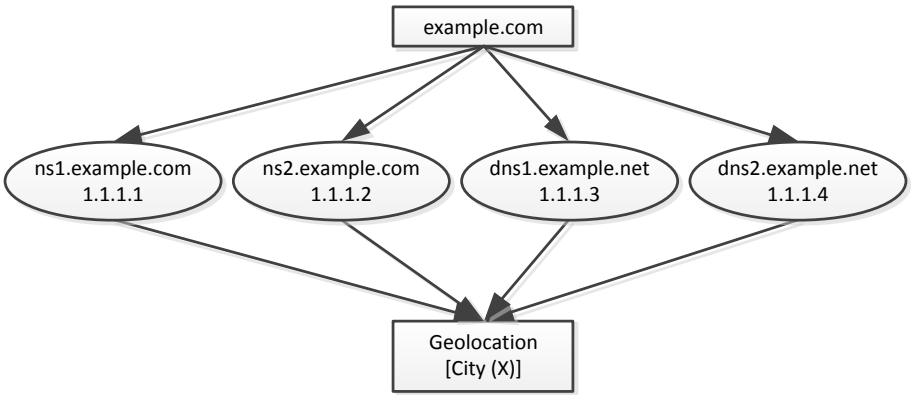

Figure 8: Geographical Location Dependency Graph of Case Study (2).

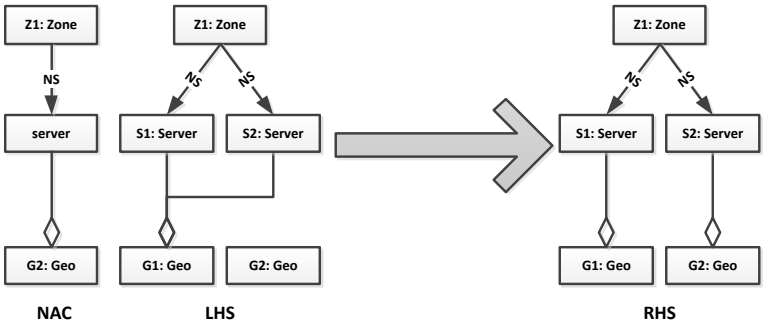

Figure 9: Refactoring Rule: MoveServerLocation.

existing efforts, DNS configuration errors are still widespread today [17], and one of the main reasons is the lack of adequate tools to help DNS operator identify and correct configuration errors in their own domains.

Many works exist on the identification of problems in software, database and networks systems. Several books have been written on smells [12] especially in the context of object-oriented programming. Marinescu [18] presented a metric-based approach to detect code smells with detection strategies. Alikacem and Sahrouri [2] proposed a language to detect violations of quality principles and smells in object-oriented systems. Mens and Tourwe [19] have conducted a comprehensive survey of software refactoring. While software refactoring has been started with the restructuring of programs, the focus of research has extended to model refactoring [5].

7 Conclusion and Future Work

Currently, there is little consensus on the right measures and acceptable performance levels for the DNS as a whole related to availability, security, stability and resiliency. Individual operators and independent researchers have measured various aspects of the DNS, but to date little progress has been made in defining and implementing standard, system-wide metrics or acceptable service levels. Efforts to improve risk management related to DNS security, stability and resiliency must be guided by an improved ability to measure these characteristics and assess the utility of programs and resources investments. A key enabler of improving this situation will be to ensure that composite parts of DNS operations are correctly

configured, deployed, instrumented and measured.

The method presented in this paper will lay the basis for developing a visual advisory tool for system administrators to analyse, discover, and remedy operational bad smells. The diagnostic tool will consider several properties and metrics from the DNS operational model presented in this research in relation to the domain name whose zone is being modified. The tool, in a systematic process, can automatically direct the zone administrator to places in the zone file that contain potential design and deployment problems that may compromise availability, resiliency or security of a domain name before the changes become into production. Zone administrator will be able to run several scenarios and apply several refactoring rules through the tool to determine the solution that best meets their local policies.

The tool is being designed to cope with zones with very large size and need to be fast enough to be practically applicable. A set of consistent refactoring steps will be applied (or recommended) as graph transformation rules using available tools and techniques. The rule-based behaviour preservation [6] will be verified to make sure the suggested rules preserve the system functionality and improve its external qualities. Execution of the refactoring rules may introduce complex sequence of operations to transform the model changes into physical resources relocation. In order to implement some of these refactoring rules, we need to take into consideration access control permissions and physical access to, or coordination actions such as service level agreements (SLAs), with external sites. These concerns will be tackled as part of the refactoring execution steps and available techniques and tools [3] are being currently investigated.

References

[1] Working Group 4 (2012): *FINAL Report: DNS Best Practices*. Technical Report, The Communications Security, Reliability and Interoperability Council III. Available at `http://transition.fcc.gov/bureaus/pshs/advisory/csric3/CSRICIII_9-12-12_WG4-FINAL-Report-DNS-Best-Practices.pdf`.

[2] E.-H. Alikacem & H.A. Sahraoui (2009): *A Metric Extraction Framework Based on a High-Level Description Language*. In: *Source Code Analysis and Manipulation, 2009. SCAM '09. Ninth IEEE International Working Conference on*, pp. 159–167, doi:10.1109/SCAM.2009.27.

[3] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause & Gabriele Taentzer (2010): *Henshin: advanced concepts and tools for in-place EMF model transformations*. In: *Model Driven Engineering Languages and Systems*, Springer, pp. 121–135, doi:10.1007/978-3-642-16145-2_9.

[4] D. Barr (1996): *RFC 1912: Common DNS operational and configuration errors*. *International Engineering Task Force, Status: Standard*. Available at `http://www.ietf.org/rfc/rfc1912.txt`.

[5] Enrico Biermann, Karsten Ehrig, Christian Köhler, Günter Kuhns, Gabriele Taentzer & Eduard Weiss (2007): *EMF model refactoring based on graph transformation concepts*. *Electronic Communications of the EASST* 3. Available at `http://journal.ub.tu-berlin.de/index.php/eceasst/article/view/34`.

[6] Dénes Bisztray, Reiko Heckel & Hartmut Ehrig (2008): *Verification of Architectural Refactorings by Rule Extraction*. In: *Fundamental Approaches to Software Engineering*, Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 347–361, doi:10.1007/978-3-540-78743-3_26.

[7] Emiliano Casalicchio, Marco Caselli, Alessio Coletta, Salvatore Di Blasi & IgorNai Fovino (2012): *Measuring Name System Health*. In Jonathan Butts & Sujeet Shenoi, editors: *Critical Infrastructure Protection VI*, *IFIP Advances in Information and Communication Technology* 390, Springer Berlin Heidelberg, pp. 155–169, doi:10.1007/978-3-642-35764-0_12.

[8] R. Chandramouli & S. Rose (2005): *An integrity verification scheme for DNS zone file based on security impact analysis*. In: *Computer Security Applications Conference, 21st Annual*, pp. 10 pp.–321, doi:10.1109/CSAC.2005.9.

[9] Microsoft Corporation (2001): *Microsoft Responds to DNS Issues*. Technical Report, Microsoft Corporation. Available at `http://www.microsoft.com/en-us/news/press/2001/jan01/01-24dnspr.aspx`.

[10] C. Deccio, J. Sedayao, K. Kant & P. Mohapatra (2010): *Measuring Availability in the Domain Name System*. In: *INFOCOM, 2010 Proceedings IEEE*, pp. 1–5, doi:10.1109/INFCOM.2010.5462270.

[11] Bush R. Bradner S. Elz, R. & M. Patton (1997): *RFC 2182: Selection and Operation of Secondary DNS Servers*. *International Engineering Task Force, Status: Standard*. Available at `http://www.ietf.org/rfc/rfc2182.txt`.

[12] Martin Fowler (1999): *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Boston, MA, USA.

[13] A. Herzberg & H. Shulman (2013): *DNSSEC: Security and availability challenges*. In: *Communications and Network Security (CNS), 2013 IEEE Conference on*, pp. 365–366, doi:10.1109/CNS.2013.6682730.

[14] Andrew J. Kalafut, Craig A. Shue & Minaxi Gupta (2008): *Understanding Implications of DNS Zone Provisioning*. In: *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement*, IMC '08, ACM, New York, NY, USA, pp. 211–216, doi:10.1145/1452520.1452546.

[15] Verisign Labs (2012): *Trabnsitive Trust Portal*. Technical Report, Verisign Company. Available at `http://trans-trust.verisignlabs.com`.

[16] M. Lotter (1987): *Rfc 1033: Domain Administrators Operations Guide*. Available at `http://www.ietf.org/rfc/rfc1033.txt`.

[17] Keyu Lu, Kaikun Dong, Cuihua Wang & Haiyan Xu (2014): *DNS configuration detection model*. In: *Systems and Informatics (ICSAI), 2014 2nd International Conference on*, IEEE, pp. 613–618, doi:10.1109/ICSAI.2014.7009359.

[18] R. Marinescu (2004): *Detection strategies: metrics-based rules for detecting design flaws*. In: *Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on*, pp. 350–359, doi:10.1109/ICSM.2004.1357820.

[19] T. Mens & T. Tourwe (2004): *A survey of software refactoring*. *Software Engineering, IEEE Transactions on* 30(2), pp. 126–139, doi:10.1109/TSE.2004.1265817.

[20] Paul Mockapetris (1987): *RFC 1034: Domain names: concepts and facilities*. Available at `http://www.ietf.org/rfc/rfc1034.txt`.

[21] Paul Mockapetris (1987): *RFC 1035: Domain namesimplementation and specification*. Available at `http://www.ietf.org/rfc/rfc1035.txt`.

[22] William F. Opdyke (1992): *Refactoring Object-oriented Frameworks*. Ph.D. thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA. UMI Order No. GAX93-05645.

[23] Eric Osterweil, Danny McPherson & Lixia Zhang (2011): *Operational implications of the DNS control plane*. *IEEE Reliability Society Newsletter*. Available at `http://rs.ieee.org/images/files/newsletters/2011/2_2011/Operational%20Implications%20of%20the%20DNS%20Control%20Plane.pdf`.

[24] Vasileios Pappas, D. Wessels, D. Massey, Songwu Lu, A. Terzis & Lixia Zhang (2009): *Impact of configuration errors on DNS robustness*. *Selected Areas in Communications, IEEE Journal on* 27(3), pp. 275–290, doi:10.1109/JSAC.2009.090404.

[25] Venugopalan Ramasubramanian & Emin Gün Sirer (2005): *Perils of transitive trust in the domain name system*. In: *Proceedings of the IMC '05, 2005 Internet Measurement Conference*, USENIX Association, pp. 379–394. Available at `http://static.usenix.org/event/imc05/tech/full_papers/ramasubramanian/ramasubramanian.pdf`.

[26] Duane Wessels, Marina Fomenkov, Nevil Brownlee & kc claffy (2004): *Measurements and Laboratory Simulations of the Upper DNS Hierarchy*. In Chadi Barakat & Ian Pratt, editors: *Passive and Active Network Measurement*, *Lecture Notes in Computer Science* 3015, Springer Berlin Heidelberg, pp. 147–157, doi:10.1007/978-3-540-24668-8_15.