TradeMaker: Automated Dynamic Analysis of Synthesized Tradespaces

Hamid Bagheri[*][§]
hb2j@virginia.edu

Chong Tang[§]
ctang@virginia.edu

Kevin Sullivan[§]
sullivan@virginia.edu

[*]Computer Science Dept.
George Mason University
Fairfax, VA 22030 USA

[§]Computer Science Dept.
University of Virginia
Charlottesville, VA 22903 USA

ABSTRACT

System designers today are focusing less on *point* solutions for complex systems and more on *design spaces*, often with a focus on understanding tradeoffs among non-functional properties across such spaces. This shift places a premium on the efficient comparative evaluation of non-functional properties of designs in such spaces. While static analysis of designs will sometimes suffice, often one must *run* designs dynamically, under comparable loads, to determine properties and tradeoffs. Yet variant designs often present variant interfaces, requiring that common loads be specialized to many interfaces. The main contributions of this paper are a mathematical framework, architecture, and tool for specification-driven synthesis of design spaces and common loads specialized to individual designs for dynamic tradeoff analysis of non-functional properties in large design spaces. To test our approach we used it to run an experiment to test the validity of static metrics for object-relational database mappings, requiring design space and load synthesis for, and dynamic analysis of, hundreds of database designs.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms

Design, Experimentation, Performance

Keywords

Specification-driven Synthesis, Tradeoff Space, ORM, Static Analysis, Dynamic Analysis

1. INTRODUCTION

Producing systems that achieve acceptable tradeoffs among non-functional properties remains a major engineering problem. To address it, engineers are now focusing less on point solutions and more on specifying, populating, and analyzing points and regions in design spaces. The contribution of this work is an approach to specification-driven synthesis of both design spaces and common loads for fair, comparative, dynamic analysis of non-functional properties of variant designs across such spaces. A key challenge is to synthesize test loads for fair comparative analysis, because doing so generally requires specialization of common loads to the variant interfaces of variant designs. Our goal is to synthesize both design spaces and such loads automatically from common, formal, design-space specifications, to enable specification-driven automated dynamic analysis of tradeoffs among non-functional properties across large design spaces.

This paper provides one solution to this problem, particularly for design space models expressible within a relational logic [20]. Section 2 presents object-relational mapping (ORM) as a concrete driving problem. Section 3 presents a mathematical and solution framework. Section 4 details an implementation architecture using a relational constraint language (Alloy) and solver (Alloy Analyzer) for expression and synthesis of design spaces and loads. Section 5 presents an experiment using our dynamic analysis approach to test the validity of static predictors of non-functional properties of database applications. The rest of the paper presents our evaluation of this work, related work, and conclusions and thoughts about future work.

2. DRIVING PROBLEM

While our long-term aim is to improve engineering productivity and quality through advances in design space science and technology, our short-term research strategy is to use the analysis of spaces of object-relation mappings (ORM) as a tractable and useful *driving problem.*

In this domain, one starts with an object model as in Figure 1 and eventually selects one of many possible strategies for mapping such a model to a relational database, with tradeoffs in response time, space usage, and evolvability. Figure 2 illustrates two such strategies; and Listing 1, a database set-up script derived from one of these solutions.

Simple ORM solutions, many in everyday use, lock one into either point solutions or highly constrained solution spaces. To address this problem, our earlier work [6] presented a capability to synthesize comprehensive ORM design spaces from formal object model specifications. Given such a space, the challenge is to develop, validate, and apply non-functional property prediction (*analysis*) functions to designs in the space to predict properties of designs in, and tradeoffs across, the space.

ICSE '14, May 31–June 7, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2756-5/14/05 ...$15.00.

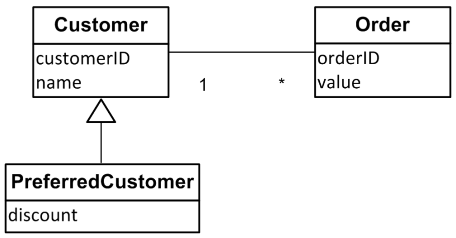


Figure 1: A simple object model with three classes, *Order*, *Customer*, and *PreferredCustomer*, a one-to-many association between *Customer* and *Order*, and with *PreferredCustomer* inheriting from *Customer*.

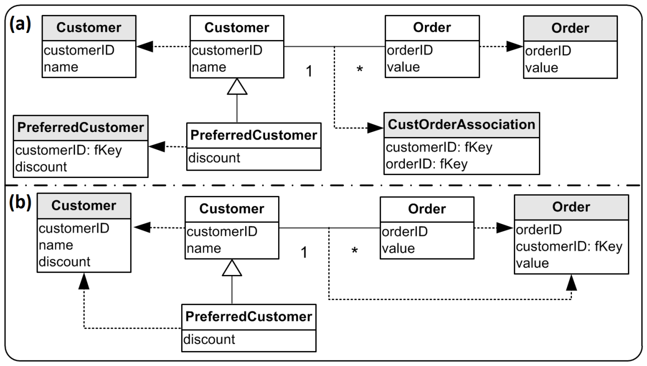


Figure 2: Two mapping strategies. White boxes represent classes; gray titles, corresponding tables, and black and white arrows, mapping and inheritance relationships. Foreign keys are marked as *fKeys*.

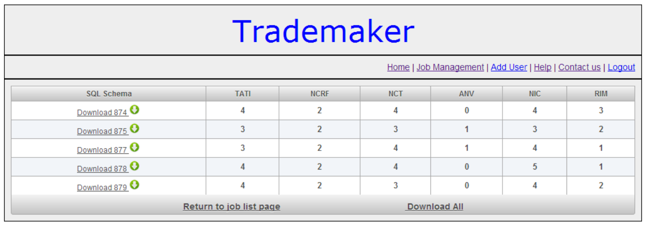
Ideally, one has a vector of easily computed, well validated analysis functions. In that case, mapping this vector of functions (or, equivalently, this vector-valued function) across the points in the space yields a multi-dimensional, non-functional property *image* of the design space. Tradeoffs, Pareto-optimal solutions, and other critical information can then be read from the results.

Static analysis functions predict properties from the structures of design representations. Such functions are often efficient, but might not be available, validated, or predictive. This point is clear in our earlier work. Cognizant of the issues, we applied published but not well validated static metrics [7,18] to our synthesized ORM spaces. The results were interesting and promising and allowed us to make progress in our research, but it was clear that questions remained.

```
CREATE TABLE ‘Order‘ (
 ‘orderID‘ int(11) NOT NULL,
 ‘orderValue‘ int(11),
 ‘customerID‘ int(11),
KEY ‘FK_customerID_idx‘ (‘customerID‘),
PRIMARY KEY (‘orderID‘)
 );

CREATE TABLE ‘Customer‘ (
 ‘DType‘ varchar(31),
 ‘discount‘ int(11),
 ‘customerID‘ int(11) NOT NULL,
 ‘customerName‘ varchar(31),
PRIMARY KEY (‘customerID‘)
 );
```

Listing 1: Synthesized MySQL database creation script elided for space and readability.


Trademaker
Home | Job Management | Add User | Help | Contact us | Logout
Return to job list page
Download All

SQL Schema	TATI	NCRF	NCT	ANV	NIC	RIM
Download 874	4	2	4	0	4	3
Download 875	3	2	3	1	3	2
Download 877	3	2	4	1	4	1
Download 878	4	2	4	0	5	1
Download 879	4	2	3	0	4	2

Figure 3: A view of our tool to provide decision-makers with Pareto-optimal OR mapping solutions based on static analysis results; columns and rows represent metrics and solution alternatives, respectively.

Can design decision makers believe such metrics? What are the actual properties and tradeoffs?

These questions took on added urgency with our production, reported here for the first time, of a web-accessible tool that implements our ORM synthesis and analysis approach. We call it Trademaker-ORM[1] (T-ORM). It supports automated, specification-driven synthesis of ORM design spaces and static analysis using the aforementioned metrics. It provides a web interface, user account and job management (job submission, asynchronous execution, status reporting, persistence), computation and presentation of Pareto-optimal subsets of synthesized designs under the given metrics, and synthesis of SQL databases for selected designs. Figure 3 presents a screen shot of a T-ORM run. Rows present Pareto-optimal designs, and columns, analysis results. Listing 1 presents an SQL database creation script obtained by selecting a design from the table.

To test the validity of the results that T-ORM reports, we turned to dynamic analysis. The combination of ample computing capacity and our ability to synthesize many running databases for given object models suggested that we measure properties and tradeoffs of actual running systems, in the spirit of what Cadar et al. call *multiplicity computing* [11]. Doing so could at least validate the static metrics through statistical analysis of the power of these metrics to predict dynamically measured results. If we could validate the static metrics, we could use them for efficient analysis. If not, we could fall back on less efficient but more trustworthy dynamic analysis.

The problem was now to figure out how to automate *fair* comparative dynamic analysis of diverse database designs. There are of course many commercial tools for generating database testing loads from schemas. In our case, however, many variant schemas implement a common object model. An application that operates against an object model will create what we will call an *abstract* load that any implementations would have to handle. The translation from abstract load to concrete operations on a database would be implemented by an application based on a single choice of mapping strategy. We faced the need to synthesize hundreds or thousands of such load specialization functions.

3. ALGEBRAIC MODEL

The insight that enabled such synthesis was that we could recover, from synthesized database designs, abstraction functions relating designs back to the object model specifications from which they were derived, and that from these

[1]www.jazz.cs.virginia.edu:8080/Trademaker

abstraction functions we could derive functions for concretizing abstract loads synthesized from the same object models. The commutative diagram in Figure 4 presents the resulting mathematical structure. $Design_{abst}$ is a set of formal design space specifications in a particular domain, inductively defined by the grammar and semantics of the language in which the models are specified. In work to date, we represent design space specifications as expressions in a domain specific language embedded in a relational logic.

$$\begin{tikzcd}
Design_{abst} \arrow[rr, "l"] \arrow[dd, shift right, "c"'] & & Load_{abst} \arrow[dd, "t"] \\
& \arrow[phantom, "\xRightarrow{compute}"] & \\
Design_{conc} \arrow[uu, shift right, "a"'] \arrow[rr, dashed, "cl"] & & Load_{conc}
\end{tikzcd}$$

Figure 4: Algebraic structure of the Trademaker approach.

To an abstract model, $m \in Design_{abst}$, we apply a design space synthesis (concretization) function, c, to compute $c(m) \subset Design_{conc}$, the space of concrete design variants from which we want to choose a design to achieve desirable tradeoffs. Relation c can be seen as mapping abstract, intensional models of design spaces to extensional representations, namely sets of concrete design variants. We represent the design space synthesis function, c, as a semantic mapping predicate in our relational logic, taking expressions in the abstract modeling language to corresponding concrete design spaces. Relation a, an abstraction relation, explains how any given concrete design, $d \in c(m)$, instantiates (i.e., is a logical model of) its abstract model, m. Function c is specified once for any given abstract modeling language, as a semantic mapping predicate in our relational logic.

Relation l, an abstract load generation relation, similarly maps an abstract model, $m \in Design_{abst}$, to a set of abstract loads $l(m) \subset Load_{abst}$.

To dynamically analyze a concrete design, $d \in c(m)$, an abstract load, $l_d \in l(m)$ has to be specialized for the particular design. The function t, a load concretization function, serves this purpose. We compute t from a. As long as $c(m)$ preserves a representation of a in its output, then from any single design space model, m, we can synthesize a concrete design space, and both abstract and concretized loads.

The derivation of t from a induces a mapping, cl, from concrete designs to concrete loads parameterized by a choice of abstract load. This function completes the commutative diagram. We do not actually implement this mapping. The next section describes instantiation of our approach for the particular domain of object-oriented relational database persistence mappings.

4. MODEL IMPLEMENTATION

This section presents an implementation architecture for our approach (for ORM). Figure 5 gives a high-level overview. Boxes represent processing modules, and ovals, module inputs and outputs. Abstract models are given as expressions

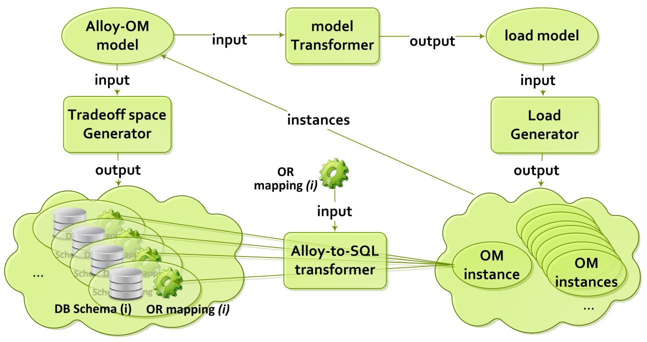


Figure 5: High-level overview of the TradeMaker implementation for the particular domain of ORM.

in Alloy-OM, a domain-specific language embedded in Alloy's relational logic language. Roughly speaking, we specify c and l as predicates in Alloy. Conjoining these predicates to an Alloy-OM model yields a specification of the desired output space. The Alloy analyzer computes the results, encoded as Alloy models, which we then unparse into useful forms. From concrete design models we extract SQL database creation scripts. From abstract load models combined with choices of concrete databases, we derive concrete loads represented as sequences of SQL insert and select queries.

The rest of this section details how we implement the main components of the approach, corresponding to functions c, l, and t. Subsection 4.1 explains how we implement c, for synthesizing concrete models from abstract design space specifications; 4.2, how we implement l, for synthesizing abstract loads from the same specifications; and 4.3, how we implement t, for concretization abstract loads.

4.1 Design Space Synthesis

Mapping abstract object models to concrete designs was the subject of earlier work [6]. Novel results since that work include substantial refinement and validation of mapping rules and the production of functions that unparse concrete designs representations, encoded as Alloy objects, into SQL database creation scripts. This translation is important for tool users as a key to automating dynamic analysis.

To make this paper self-contained, this subsection briefly reviews the approach. We then briefly describe relevant new work. Inspired by a bottom-up model-driven approach [5], we start by expressing object models in a domain-specific language embedded in Alloy, called Alloy-OM. We then use Alloy's constraint solver to synthesize concrete OR mapping strategies, realizing the function, c. The synthesizer uses our Alloy-encoded formalizations of best practices for object-relational mapping, described informally in the literature [10, 21, 25].

Alloy-OM supports three main constructs: *Class, Attribute* and *Association*. Each class in an (e.g., UML) object model appears as a *Class* signature in Alloy-OM. Each attribute of a given class appears as an Alloy-OM *Attribute* in the *attribute set* of the corresponding Alloy-OM Class signature. Each association in the object model appears as an Alloy-OM *Association* signature.

Listing 2 presents a fragment of an Alloy-OM model for our customer-order example. The `Order` class has two attributes, "orderID" and "orderValue", assigned to the "attrSet" field of the Order class. The id field specifies the

```
1  one sig Order extends Class{}{
2    attrSet = orderID + orderValue
3    id=orderID
4    isAbstract = No
5    no parent
6  }
7  one sig CustomerOrderAssociation extends
       Association{}{
8    src = Customer
9    dst = Order
10   src_multiplicity = ONE
11   dst_multiplicity = MANY
12 }
```

Listing 2: Order class in Alloy-OM.

orderID as the identifier of this class. The last two lines of the Order signature specification denote that Order is not an abstract class and has no parent. Lines 7–12 then specify the `CustomerOrderAssociation`, along with the number of object instances that can be at each end of the association, denoted as *src_* and *dst_multiplicity*. Alloy-OM is not a sophisticated modeling language. The salient point is that embedding it in Alloy allows us to use Alloy relations to encode, and the Alloy analyzer to compute, formally specified semantic mapping rules to other domains: here concrete database designs and abstract loads for those designs.

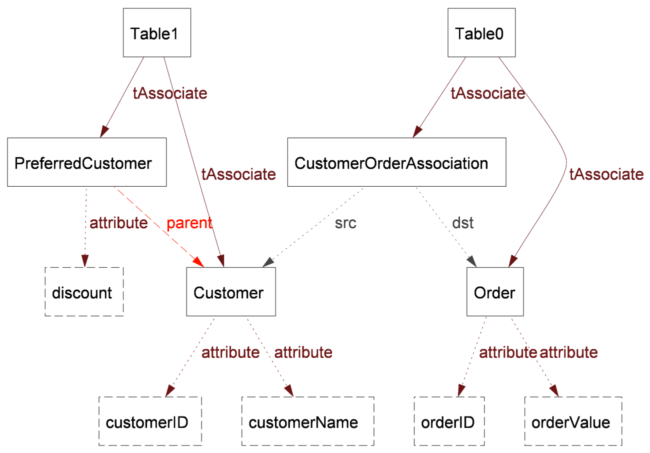


Figure 6: OR mapping for customer-order example

Figure 6 presents a graphical depiction of an Alloy object encoding a synthesized OR mapping solution. This solution is one of five Pareto-optimal solutions in the design space for our customer-order object model. The diagram is accurate but edited to omit some details for readability. In this diagram, Table1 is associated to Customer and PreferredCustomer classes, and Table0 is associated to both Order and CustomerOrderAssociation.

From this Alloy solution, our tools generate the SQL script of Listing 1. The script sets up a database with the two tables: *Order*, with attributes *orderID*, *customerID* and *orderValue*; and *Customer*, with attributes, *customerID*, *customerName*, *discount*, and *DType*. Both *Customer* and *PreferredCustomer* objects are stored in this table under this particular mapping strategy, with the *DType* field distinguishing the type of record stored.

4.2 Abstract Load Synthesis

Our approach to synthesizing abstract loads starts with the automated transformation of a given Alloy-OM model

```
1  fact {
2    all o1,o2:CustomerOrderAssociation|o1.orderID =
        o2.orderID and o1.customerID = o2.customerID
         => o1=o2
3  }
4
5  fact {
6    all o:CustomerOrderAssociation| one c:Order| o.
        orderID = c.orderID
7    all o:CustomerOrderAssociation| one c:Customer|
        o.customerID = c.customerID
8  }
```

Listing 4: Part of the load model specification generated for customer-order association.

into a related Alloy specification that we call a load model. We then use the Alloy Analyzer to synthesize abstract loads from this load model. Alloy solutions to the load model encode abstract object model data instances (*OM-instances*), which are what we take as synthesized loads with which to test synthesized designs. This section describes this functionality in more detail.

For each instance of *Class* and *Association* in the Alloy-OM model, our *model transformer* synthesizes a signature definition. When the class under consideration inherits from another class, the synthesized signature definition extends its parent signature definition. Given the specification of *Order* represented in Listing 2, the following code snippet represents its counterpart in a synthesized load model.

```
sig Order{
  orderValue: one Int,
  orderID: one Int
}
```

The *one* multiplicity constraints used in the declaration of elements' signatures within the Alloy-OM model (Listing 2) specify them as singleton signatures. While these constraints are required by the tradeoff space generator (e.g. to not generate multiple tables for a class in the model), they are unneeded for load generation, and thus omitted in the load model. The element attributes in the object model are also declared as fields of the corresponding load signature definition representing relations from the signature to the attribute *type*.

Finally, two sets of constraints are synthesized as *fact* paragraphs in the load model to guarantee both *referential integrity* of generated data as well as *uniqueness* of element identifiers with reference to the set of element instances to be generated. Referential constraints require every value of a particular attribute of an element instance to exist as a value of another attribute in a different element.

Consider the association relationship between Customer and Order classes from our running example (Figure 2). The code snippet of Listing 4 represents synthesized constraints in the load model for the customer-order association.

The expression of lines 1–3 states that if any two elements of type *CustomerOrderAssociation* have the same *orderID* and *customerID*, the elements are identical. This constraint rules out duplicate elements. The fact constraint of lines 5–8 states that for any orderID and customerID fields of a *CustomerOrderAssociation*, there are *Order* and *Customer* instances with the same *orderID* and *customerID*.

Applying the Alloy Analyzer to the derived load model yields the desired load in the form of object model data instances (OM-instances). Figure 7 depicts a generated OM-instance, an Alloy solution object. This solution represents

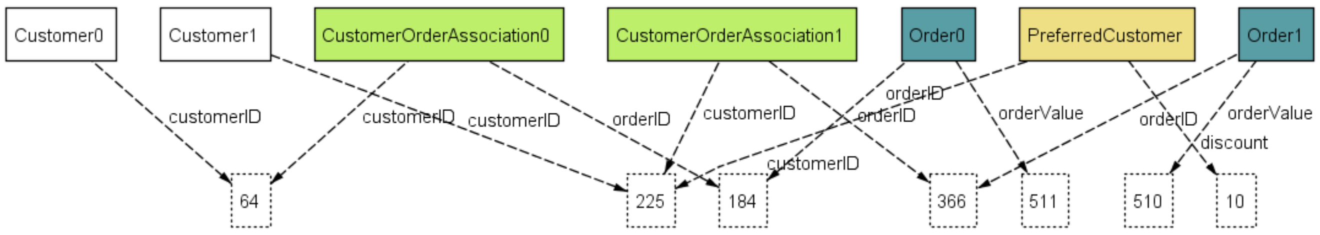


Figure 7: An example of OM-instance.

two customers with customerID of 64 and 225, the latter a preferred customer with 10 percent discount, along with their orders. From many such solutions we derive an abstract (application-object-model-level, rather than concrete-database-schema-level) load with which to test the performance of many database instances.

Improving the efficiency of the load generator. One of the challenges we faced involved the scalability of this approach to load synthesis. A large number of solutions generated by the Alloy Analyzer were symmetric to previously generated instances, and thus did not contribute usefully to the load being generated. We explored a number of ways to improve efficiency of the load generator. The one that we found worked best is the iterative refinement of the load model by adding constraints that eliminate permutations of the already generated OM-instances. Without this improvement, it took 21 hours for Trademaker-ORM to generate test loads for one of our experiments. Given this approach, the time was reduced to about 2 hours—an order of magnitude speed up in the synthesis of test loads.

4.3 Abstract Load Concretization

The next challenge we discuss is to convert abstract load OM-instance objects into concrete SQL queries on a per-database basis. This is the task of specializing abstract load elements to the variant schemas presented by different solutions in the design space. Our *Alloy-to-SQL transformer* handles this task. To create SQL statements for a given database, *Alloy-to-SQL transformer* requires an OR mapping, the abstraction function describing how that concrete database schema implements the abstract object model.

Algorithm 1 outlines this transformation. For brevity, and because it suffices to make our point, this section focuses on *insert* queries. The approach supports the generation of select and update queries as well, which are important, of course, for comprehensive dynamic analysis.

The logic of the algorithm is as follows. Iterate over all elements in a given OM-instance (e.g., classes and associations) whose values can be populated into databases through insert statements. Looks up the mapping to determine the table in which the element values should be stored. For each relational field in the associated table, if the OM-instance contains a value corresponding to that field, insert the value into the field. Otherwise, in the case that the field is a *DType*, insert the name of element into the field. Finally, if the field is a foreignKey, find the associated attribute from a relevant association in the given OM-instance, and insert its value into the field.

Consider the database alternative for our running example, in which we store the customer-order association data into the order table (Figure 2b). In that case, the field of *customerID* in the *Order* table is a foreignKey, and its values come from the associated customerOrderAssociation element.

Algorithm 1: Generate SQL Insert Statements

```
   Input: omi: OM-instance, map: OR mapping
   Output: A set of SQL insert statements
1  for element in omi do
2      T = map.TableAssociat(element);
3      F = T.fields;
4      for field in F do
5          value = getValueFromOMI(field);
6          if value != null then
7              add “field = value” into statements
8          end
9          else
10             if field == “DType” then
11                 value = element.name;
12             end
13             if isForeignKey(field) then
14                 attr = findAttributeFromAssociation(field);
15                 value = getValueFromOMI(attr);
16             end
17             add “field = value” into statements
18         end
19     end
20 end
```

```
INSERT INTO ‘Customer‘ (‘customerID‘,‘DTYPE‘)
    VALUES (64,'Customer');
INSERT INTO ‘Customer‘ (‘customerID‘,‘DTYPE‘)
    VALUES (225,'PreferredCustomer');
INSERT INTO ‘Order‘ (‘orderID‘,‘orderValue‘,‘
    customerID‘) VALUES (184,511,64);
INSERT INTO ‘Order‘ (‘orderID‘,‘orderValue‘,‘
    customerID‘) VALUES (366,510,225);
```

Listing 5: Generated SQL insert statements from OM-instance of Figure 7 for implementation mapping of Figure 6.

Listings 5 represents the set of SQL insert statements generated from the OM-instance of Figure 7 according to the mapping of Figure 6. The first two generated statements define insert queries to store instances of Customer and PreferredCustomer into the Customer table along with appropriate *DType* values for each one. The next two statements then store instances of Order and CustomerOrderAssociation into the Order table.

5. DYNAMIC ANALYSIS EXPERIMENT

As an experimental test of our approach to specification-driven, automated dynamic analysis of non-functional property tradeoffs across design spaces, we apply the approach to test the validity of the static predictors of database performance. We formulate, test, and provide experimental data in support of three driving hypotheses:

- H1: The ordering of alternatives predicted by the static metrics predicts that of the dynamic analysis results

- H2: The relative magnitudes of static measures of alternatives predict those of the dynamic analysis results
- H3: Dynamic analysis using scale-limited synthesized loads predicts performance under much larger loads

This section summarizes the design and execution of our experiment, the data we collected, its interpretation, and our results, which include novel findings regarding these metrics.

5.1 Static Metrics Suite

The choice of mapping strategy impacts key non-functional system properties. Response time performance, storage space and maintainability are among the set of quality attributes defined by the ISO/IEC 9126-1 standard that are influenced by the choice of OR mappings. To statically measure these attributes in an ORM design space, we use a set of metrics suggested by Holder et al. [18] and Baroni et al. [7]. The metrics are called Table Access for Type Identification (TATI), Number of Corresponding Table (NCT), Number of Corresponding Relational Fields (NCRF), Additional Null Value (ANV), Number of Involved Classes (NIC) and Referential Integrity Metric (RIM). In this work, we focus on three of these metrics for time and space performance. Maintainability is out of scope as we cannot measure it using dynamic analysis technique.

5.1.1 Table Accesses for Type Identification (TATI)

Table Accesses for Type Identification (TATI) is a performance metric for polymorphic queries [18]. According to the definition, given a class C, TATI(C) defines the number of different tables that correspond to C and all its subclasses. Our tools total up TATI values for each class as the overall TATI measure for each solution alternative.

5.1.2 Number of Corresponding Tables (NCT)

Number of Corresponding Tables (NCT) is a performance metric for insert and update queries. This metric specifies the number of tables that contain data necessary to assemble objects of a given class [18]. According to the definition, given a class C, NCT(C) equals to NCT of its direct super class, if C is mapped to the same table as its super class. Otherwise, if C is mapped to its own table, NCT(C) equals to NCT of its direct super class plus one. Finally, if C is a root class, NCT(C) equals to 1. Our tool computes totaled NCT values over classes as the NCT measure for each solution alternative.

5.1.3 Additional Null Value (ANV)

The Additional Null Value (ANV) metric specifies the storage space for null values when different classes are stored in a common table [18]. According to the definition, given a class C, ANV(C) equals to the number of non-inherited attributes of C multiplied by the number of other classes that are being mapped to the particular table to which C is mapped. Our tools present totals for ANV values over all classes as the ANV measures for each solution alternative.

5.2 Static Analysis of Synthesized Designs

To apply these metrics to synthesized solutions, we designed specific Alloy queries. Here we describe one for measuring the TATI metric. The others are evaluated similarly.

$$TATI(C) = \#(C.*(\sim parent).\sim tAssociate)$$

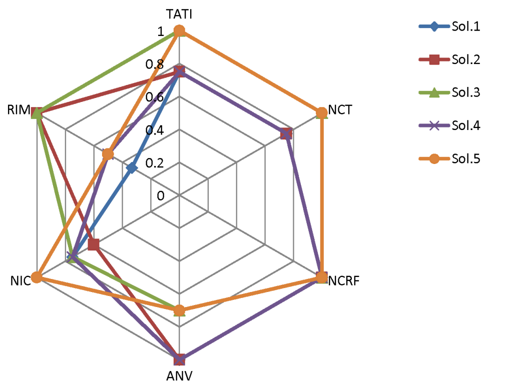


Figure 8: Multi-dimensional *quality measures* for pareto-optimal solutions.

Here the dot operator denotes a relational join. The Alloy $\sim$ operator represents the transpose operation over a binary relation, which reverses the order of atoms within the relation. Given the *tAssociate* (abstraction) relation that maps tables to their associated elements (i.e. Class or Association) within the object model, its transpose is the relation that maps each element to its associated table within the relational structure. The Alloy * operator represents the reflexive-transitive closure operation of a relation. Accordingly, the expression of "*C.*(∼parent)*" states a set of classes that have the class "C" as their ancestor in their inheritance hierarchy. The query expressions then, by using the Alloy set cardinality operator #, computes the TATI metric.

Our static metrics suite comprises six such static measures. The vector of these functions defines a 6-dimensional static analysis function applicable to Alloy-synthesized concrete designs (e.g., Figure 6). Our tools map this function over all elements of a synthesized design space to produce a tradeoff surface. The spider diagram, shown in Figure 8, illustrates one Pareto-optimal point on that surface for our example customer-order system. To display quality measures in one diagram, we normalized the values. Such diagrams can assist in conducting tradeoff analyses by making it easier to visualize and compare alternatives. According to the diagram, if the designer opts for performance, she may decide to use Sol. 5 instead of Sol. 4, as the latter has worse values for the TATI and NCT performance metrics.

Of course none of this theory or machinery is very useful if the metrics themselves are not predictive of the actual non-functional (performance) properties of candidate designs. The rest of this section presents our experiment in automated dynamic analysis, the goal of which was to help us answer this question.

5.3 Four Subject Systems

We synthesized design spaces and compared static predictions with dynamic results for four subject systems. The first is the object model of an E-commerce system adopted from Lau and Czarnecki [24]. It represents a common architecture for open source and commercial E-commerce systems. It has 15 classes connected by 9 associations with 7 inheritance relationships. The second and third object models are for systems we are developing in our lab. Decider is another system to support design space exploration. Its object model has 10 Classes, 11 Associations, and 5 inheritance relationships. The third object model is for a system, CSOS, a kind of cyber-social operating system meant to help coordinate people and tasks. In scale, it has 14 Classes, 4 Associations, and 6 inheritance relationships. We also analyzed an extended version of our customer-order example.

5.4 Planning and Execution

Our experimental procedure involved the synthesis of both design spaces of database alternatives and several abstract loads in a variety of sizes for each subject system. Given the synthesized schemas, we created a database for each alternative. We then populated generated data into databases, and ran concrete queries over those databases. We measured and collected the numbers of concrete queries generated from abstract loads for each database alternative, query execution time, as well as the size of each database.

We used an ordinary PC with an Intel Core i7 3.40 Ghz processor and 6 GB of main memory, with SAT4J as our SAT solver. Database queries were performed on a MySQL database management system (DBMS) installed on a machine equipped with an AMD Opteron 6134 800 Mhz processor and 64GB memory. Data and statistical information are available at http://jazz.cs.virginia.edu:8080/Trademaker/data.

Table 1: Design space sizes for subject systems.

Subj. Sys.	Solutions	Eq.Classes	Pareto Sols.
Decider	386	154	12
E-commerce	846	360	16
CSOS	278	121	21
Cust-order (ext)	28	14	10

Table 1 summarizes generated solution space for each subject system. There is one row for each system. The columns indicate the total number of solutions, the number of static equivalence classes where equivalence is determined by equality of static analysis results, and the number of Pareto-optimal solutions under the given static metrics.

We investigated and compared two different methods for generation of data sets. The first method generated data using our formal synthesis methods. For the second, we hand-developed a load generator for generating large loads that nevertheless respect the constraints in our object models (e.g., referential constraints between elements).

Three data sets were developed for each subject system to support the task of evaluating the static metrics.

Dataset 1. This data set is generated using the Alloy-based data generator, where the maximum bit-width for integers is restricted to 5. This leads to the generation of small data set for our experiments.

Dataset 2. This data set is generated using our Alloy-based data generator. The maximum bit-width for integers is restricted to 10, which leads to the generation a larger data set compared with the former data set.

Dataset 3. As with many formal techniques, the complexity of constraint satisfaction restricts the size of models that can practically be analyzed and synthesized [15, 22]. For experimental purposes, we hand-implemented a more scalable data generator. It does not generate queries directly, but rather replaces the constraint solver for synthesis of abstract loads. Having synthesized larger abstract loads in the form of OM-instances, using the mechanisms already used in the Alloy-based data generator (cf. 4.3), the generator then transforms abstract loads into sets of concrete queries targeting diverse implementation alternatives.

Table 2: Part of the generated data sets for the ecommerce experiment; the first row shows abstract loads generated for the ecommerce system within each data set; each cell in the other rows corresponds to the size of generated concrete load for the database alternative and data set given on the axes.

E-commerce	Dataset 1	Dataset 2	Dataset 3
abstract load	862	2,576	164,813
Sol.19 (conc. load)	456	13,698	2,471,700
Sol.121 (conc. load)	320	9,770	1,647,800
Sol.264 (conc. load)	397	12,073	2,142,140
Sol.348 (conc. load)	379	11,395	1,977,360

Table 2 presents the sizes of generated data sets for some of the solution alternatives for the E-commerce system. The number of concrete queries refined from a common abstract load is different in various solution alternatives, depending on the way that each implementation mapping alternative refines the abstract object model into the concrete representation in relational structure (cf. 4.3).

5.5 Results for Hypothesis H1 (Order)

To test the predictive power of our static metrics, we compared its predictions against the results of our dynamic analysis. To evaluate our first hypothesis—whether the relative order of implementation alternatives is predicted by static metrics—we compute Spearman correlation coefficients, an appropriate correlation statistic for order-based consistency analysis. It measures the degree of consistency between two ordinal variables [27]. A correlation of 1 indicates perfect correlation, while 0 indicates no meaningful correlation. Negative numbers indicate negative correlations.

Table 3: Correlation coefficients between the relative order of solution alternatives predicted by static metrics and those observed from actual runtime measures.

		TATI	*NCT*	*ANV*
Decider	Dataset1	0.93	0.93	-0.98
	Dataset2	0.96	0.96	-0.92
	Dataset3	0.95	0.92	-0.91
E-commerce	Dataset1	0.97	0.96	-0.95
	Dataset2	0.98	0.95	-0.95
	Dataset3	0.97	0.94	-0.95
CSOS	Dataset1	0.83	0.76	-0.97
	Dataset2	0.57	0.62	-0.79
	Dataset3	0.58	0.74	-0.55
Cust-order(ext)	Dataset1	0.89	0.92	-0.51
	Dataset2	0.74	0.53	-0.44
	Dataset3	0.66	0.82	-0.56

Table 3 summarizes correlation coefficients between static metrics and dynamic measures. The data show reasonably strong but somewhat inconsistent positive correlations between statically predicted and actual run-time performance for TATI (average correlation of 0.84) and NCT (average correlation of 0.84). These metrics appear moderately to strongly predictive of the relative ordering databases run-time performance, at least for the kinds of loads employed in our experiments.

The performance of the ANV predictor varies across the subject systems. ANV predicts well in the E-commerce and Decider experiments and moderately in the CSOS data, but weakly in the customer-order-extended set. Moreover, the results show negative correlation between the ANV metric and database size. As number of null values increases, size decreases. This observation for the ANV metric is in direct contrast to what is predicted. One possible reason is that when the ANV metric increases, the number of tables for the database solution under consideration decreases. Assuming that the database system efficiently stores null values, the database size would reduce.

To further evaluate predictiveness of static metrics, we consider the case in which designers use each static metric as a two-class classifier. We, thus, measure precision, recall and F-measure as follows:

Precision is the percentage of those alternatives predicted by a given metric as more preferable in terms of a given quality attribute that were also classified as more preferable by the actual analysis: $\frac{TP}{TP+FP}$

Recall is the percentage of alternatives classified more preferable by the actual analysis that were also predicted as more preferable by the a given metric: $\frac{TP}{TP+FN}$

F-measure is the harmonic mean of precision and recall: $\frac{2*Precision*Recall}{Precision+Recall}$

where TP (true positive), FP (false positive), and FN (false negative) represent the number of solution alternatives that are truly predicted as preferable, falsely predicted as preferable, and missed, respectively.

While static metrics output predictions of quality characteristics as natural numbers, actual analysis of query execution performance and required storage space are in terms of Seconds and Bytes, respectively. To classify an alternative as *preferable*, we thus use median for each set of result values as a threshold. We measure evaluation metrics for each subject system with respect to three data sets.

Table 4: Experimental results of evaluating OR metrics as two-class classifiers.

		TATI			*NCT*			*ANV*		
		Prec	Recall	F-measure	Prec	Recall	F-measure	Prec	Recall	F-measure
Decider	DS1	1	0.75	0.86	0.83	0.83	0.83	0	0	0
	DS2	0.83	0.83	0.83	1	1	1	0.17	0.17	0.17
	DS3	1	1	1	0.83	0.83	0.83	0.17	0.17	0.17
E-commerce	DS1	1	1	1	1	1	1	0	0	0
	DS2	1	1	1	1	1	1	0	0	0
	DS3	0.9	0.9	0.9	1	1	1	0	0	0
CSOS	DS1	0.75	0.82	0.78	0.75	0.82	0.78	0.36	0.33	0.35
	DS2	0.83	1	0.91	0.67	0.80	0.73	0.36	0.33	0.35
	DS3	0.83	1	0.91	0.83	1	0.91	0.36	0.33	0.35
Cust-order (ext)	DS1	1	1	1	1	1	1	0.43	0.60	0.50
	DS2	0.83	1	0.91	0.67	0.80	0.73	0.57	0.67	0.62
	DS3	0.83	1	0.91	0.83	1	0.91	0.43	0.60	0.50

Table 4 summarizes the results of our experiments to evaluate the accuracy of static metric predictors as two-class classifiers. The average precision, recall and F-measure are depicted in Figure 9. The results show the accuracy of the TATI and NCT metrics in classifying implementation alternatives in terms of their run-time performance. The average precision and recall for all four experiments are about 90%, showing a low rate of both false positives and false negatives. The ANV metric, however, achieves the average under 30% in all evaluation metrics.

The experimental data thus suggests that, under the generated abstract loads, the relative order of implementation alternatives predicted by static metrics of TATI and NCT is indicative of their comparative preference in actual runtime performance, but this is not the case for ANV as a static predictor of storage space.

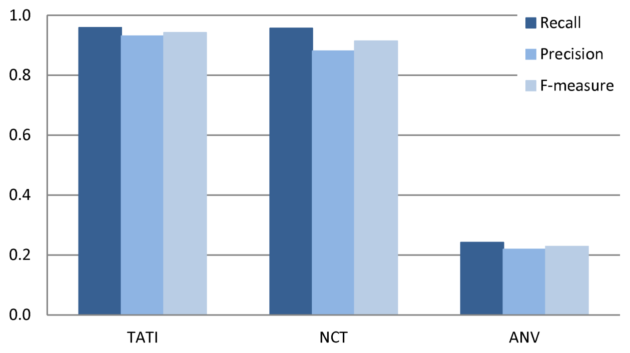


Figure 9: Bar plot of the average precision, recall and F-measure for considering static metrics as two-class classifiers.

5.6 Results for Hypothesis H2 (Magnitudes)

To address the second hypothesis—relative magnitude of static predictions matter—we employ a coefficient of determination denoted R^2, as a metric for how well actual outcomes are predicted by the static metrics. Figure 10 plots the results. For brevity, only results from *Dataset 3* are presented; other data sets give similar results.

The performance of the predictors varies widely across systems and predictors. TATI and NCT are predictive of performance for the E-commerce and Decider systems, but relatively poor predictors for CSOS. TATI performs poorly in the customer-order-extended data. ANV predicts size in the E-commerce experiment, and moderately in the Decider data, but inconsistently and weakly in the CSOS data and not at all in the customer-order-extended set.

We interpret this data as suggesting that the relative magnitudes of static metrics for various solution alternatives are not reliably indicative of the relative magnitudes of actual performance, and that ANV is a poor indicator of the storage space. One is advised to use such static metrics with caution. While TATI and NCT metrics predict the relative order of solution alternatives with high confidence, the difference in predicted values of two alternatives is not a good indicator of their actual run-time difference.

5.7 Results for H3 (Small vs. Large Loads)

To address the third hypothesis—that small, formally synthesized loads predict the outcomes of much larger loads—we employ the Pearson product-moment correlation statistic. Pearson measures the degree of linear dependence between two variables, not necessarily ordinal, as opposed to the Spearman test. A correlation of 1 represents perfect correlation, and 0, no meaningful correlation.

We summarize correlation coefficients between experimental results obtained from smaller data sets of 1 and 2 and that of the large *Dataset 3* in Figure 11. The average Pearson correlation coefficient between *Dataset*3 and the first and second data sets are 88% and 94%, respectively. These data lend support to the proposition that smaller-scale test sets produced by specification-driven synthesis can provide valid predictions of performance under larger, more realistic loads.

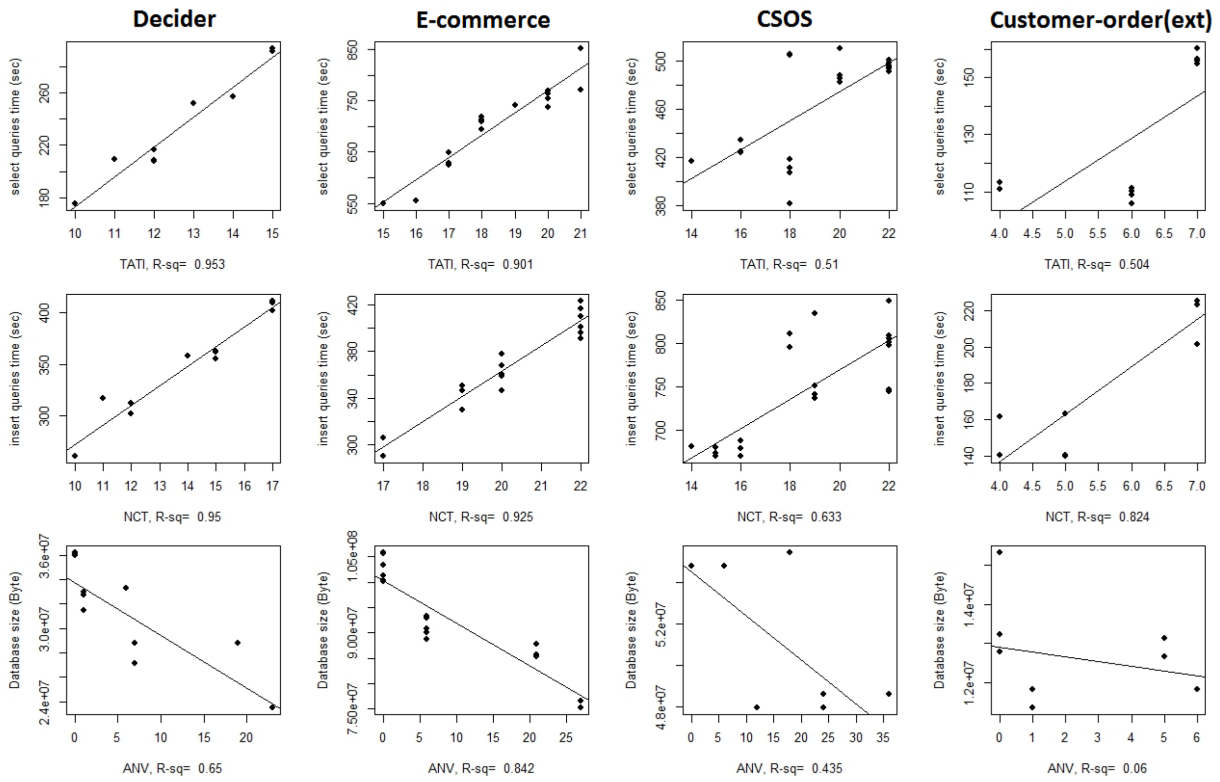


Figure 10: Correlation between static metrics and actual run-time measure; rows represent scatter plots of observed values versus predicted values by TATI, NCT and ANV metrics from top to bottom, respectively; R^2 correlation coefficient is shown at the bottom of each plot.

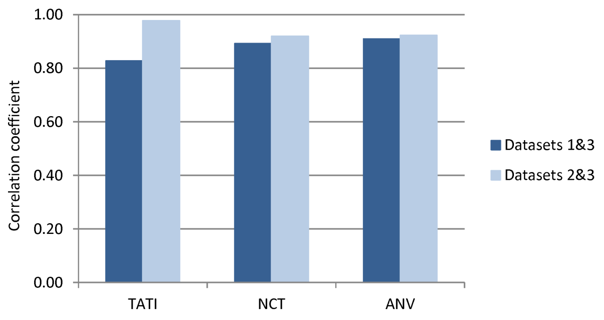


Figure 11: Summary of Pearson correlation coefficients between experimental results obtained from smaller data sets and that of the large *Dataset*3.

6. OUR EVALUATION OF THIS WORK

The overarching problem this work addresses is interesting and important: the need for improved science to support decision making in complex and poorly understood tradeoff spaces, particularly involving tradeoffs among non-functional properties, also sometimes called *ilities*. We need languages in which to specify design spaces, techniques for synthesizing and analyzing design spaces, mechanisms for mapping static and dynamic analysis functions across design spaces, techniques for validating such metrics, and tools that enable engineers use the science to improve real engineering practice [3, 4]. We also need measures for *ilities* that are important but hard to measure today: for evolvability, some dependability properties, affordability of construction, and more.

This paper contributes some new results to the science and engineering of tradeoff analysis of non-functional properties. It suggests the possibility of useful formal languages for specifying design spaces in support of formal synthesis of both designs and comparative analysis loads. We showed that specialization of common loads is enabled by access to abstraction functions from concrete to abstract designs, which can be embedded in the results of design synthesis. Concretization functions proved useful not only for scale-limited, formally synthesized loads, but for concretizing abstract loads produced by other means. We also presented an experiment using our tool to test the validity of static predictors of database performance based on published but not validated metrics. Two of the metrics appear to produce meaningful signals, while the third appears not useful. The data also indicate a need for caution in relying on the static metrics. Their predictive power, even in the "good" cases, varied across application models. That said, we can now provide automated dynamic analysis as a fall-back. We are integrating support for invoking such automated analysis into our Trademaker-ORM tool. Trademaker-ORM itself has real potential utility for object-relational mapping and partial application synthesis; but its greater significance is as a demonstration of our research results and a testbed for further research on formal tradespace modeling and analysis.

There are of course limitations in our approach and in this work. We mention those most relevant to a proper evaluation of this effort. First, the static metrics we evaluated *sum* the values of published metrics over the elements of each design alternative. We thus extended the original metrics and our statistical results should technically be read as pertaining to these extensions of the original measures.

Second, while our synthesis mechanisms are implemented and working, our infrastructure for running synthesized concrete loads against synthesized designs still relies on some manual processing. Our statistical data were thus derived by dynamic analyses of certain *subsets* of our synthesized designs. We selected the subsets deemed Pareto-optimal by the static metrics. As our infrastructure matures, we will conduct whole-space dynamic analyses, which we expect to produce results consistent the basic result presented here. We are on a path to support automated whole-space dynamic analysis through Trademaker-ORM. The work reported in this paper did nevertheless involve the synthesis and dynamic analysis of over 300 database alternatives.

Third, our experiments to date tested our hypotheses for "random" loads of varying sizes. Real applications will generally produce non-random loads. Whether the static metrics we tested are predictive for large, real applications remains unclear. On the other hand, we offer dynamic analysis at scale as an alternative. We envision a future in which some systems run many design variants in parallel, perhaps with small but representative loads abstracted from real loads on live systems, to detect conditions in which dynamic switching to new implementation strategies should be considered.

Finally, there is the issue of scalability. Using Alloy as a constraint solver entails scalability constraints. We can handle object models with tens of classes. Industrial databases often involve thousands of classes. It is unlikely that our current implementation technology will work at that scale. For now, it does have real potential as an aid to smaller-scale system development. That we can present an object model for a realistic web service, synthesize a broad space of ORM strategies, select one based on tradeoff analysis, automatically obtain an SQL-database setup script, provide it Java EE, and have much of an enterprise-type application up and running with little effort is exciting, even if it does not address (yet) the most demanding needs of industry.

7. RELATED WORK

Much work is related to this research. Numerous techniques have been developed for database test generation [8, 9, 22, 26], including the generation of realistic loads for TPC benchmarks [1]. While some use constraint solvers [22, 26], none generate common loads over spaces of alternative schemas. Doing this requires enforcement of abstract design constraints as well as constraints implied by concretization mappings for each alternative. Trademaker, to our knowledge, is the first tool with this capability.

Others have derived databases from specifications. Krishnamurthi et al. [23] mapped Alloy specifications into Scheme implementations with a focus on databases. Cunha and Pacheco [12] translated a subset of Alloy into corresponding relational operations. These research efforts share with ours the emphasis on using formal methods. Our work differs fundamentally in its emphasis on the generation of *spaces* of implementation alternatives, not just point solutions.

Much work has focused on object-relational mapping approaches to the object-relational impedance mismatch problem [10, 19, 21, 25]. Philippi [25] categorized the mapping strategies in a set of pre-defined quality trade-off levels, which are used to develop a model driven approach for the generation of OR mappings. Cabibbo and Carosi [10] discussed more complex mapping strategies for inheritance hierarchies, in which various strategies can be applied independently to parts of a multi-level hierarchy. Our approach is novel in having formalized ORM strategies previously informally described in some of these research efforts, thereby enabling automatic generation of OR mappings for each application object model.

Drago et al. [14] considered OR mapping strategies as a variation points in their work on feedback provisioning. They extended the QVT-relations language with annotations for describing design variation points, and provided a feedback-driven backtracking capability to enable engineers to explore the design space. While this work is concerned with the performance implications of choices of per-inheritance-hierarchy OR mapping strategies, it does not attack the problem that we address, the automation of dynamic analysis through synthesis of design spaces and fair loads for comparative dynamic analysis.

The other relevant thrust of research has focused on mapping UML models enriched with OCL invariants into relational structures and constraints. Heidenreich et al. [17] developed a model-driven transformation framework to map object models represented in UML/OCL into declarative query languages, such as SQL and XQuery. Badawy and Richta [2] provided some rules guiding derivation of declarative constraints and triggers from OCL specifications. Extending the same line, Al-Jumaily et al. [16] developed a model-driven tool transforming the OCL constraints into SQL triggers. Demuth et al. [13] also discussed a number of different approaches to implement OCL-to-SQL mapping, and developed a tool that transforms each OCL invariant into a separate SQL view definition. Different from these research efforts transforming an object model to a single counterpart in relational structures, Trademaker generates tradeoff spaces of object-relational mappings with focus on structural mapping alternatives, rather than transformation of integrity constraints.

8. CONCLUSION

This paper makes several contributions to the science and engineering of software-intensive systems: a mathematical and implementation architecture for formal, automated *dynamic analysis of tradeoff spaces*; a principled approach to *load concretization* for specializing common loads to large numbers of variant implementations; experimental validation of (simple derivatives of) published ORM metrics—to our knowledge the first experimental evaluation of ORM metrics; and TradeMaker-ORM, an accessible and functional tool enabling tradeoff analysis in large design spaces for the particular domain of object-relational mapping, and a testbed for ongoing research of the kind reported in this paper. This paper also contributes to our broader research program, which is increasingly focused on specifying, validating, realizing, and certifying acceptable tradeoffs among non-functional properties, which remains a research challenge of the first order.

Acknowledgements

We thank Michele Claibourn and Clay Ford of UVa StatLab for their consulting and statistical assistance. Kevin Sullivan gratefully acknowledges support from the Systems Engineering Research Center (SERC) under Contract H98230-08-D-0171. SERC is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology.

9. REFERENCES

[1] TPC benchmarks. http://www.tpc.org.

[2] M. Badawy and K. Richta. Deriving triggers from UML/OCL specification. In *Information Systems Development*, pages 305–315. Springer US, Jan. 2002.

[3] H. Bagheri and K. Sullivan. Monarch: Model-based development of software architectures. In *Proceedings of the 13th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS'10)*, pages 376–390, 2010.

[4] H. Bagheri and K. Sullivan. Pol: Specification-driven synthesis of architectural code frameworks for platform-based applications. In *Proceedings of the 11th ACM International Conference on Generative Programming and Component Engineering (GPCE'12)*, pages 93–102, Dresden, Germany, 2012.

[5] H. Bagheri and K. Sullivan. Bottom-up model-driven development. In *Proceedings of the International Conference on Software Engineering (ICSE'13)*, pages 1237–1240, 2013.

[6] H. Bagheri, K. Sullivan, and S. Son. Spacemaker: Practical formal synthesis of tradeoff spaces for object-relational mapping. In *Proceedings of the 24th International Conference on Software Engineering and Knowledge Engineering*, pages 688–693, San Francisco Bay, USA, 2012.

[7] A. L. Baroni, C. Calero, M. Piattini, and O. B. E. Abreu. A formal definition for object-relational database metrics. In *Proceedings of the 7th International Conference on Enterprise Information System*, pages 334–339, 2005.

[8] C. Binnig, D. Kossmann, E. Lo, and M. T. Özsu. QAGen: generating query-aware test databases. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 341–352, New York, NY, USA, 2007. ACM.

[9] N. Bruno, S. Chaudhuri, and D. Thomas. Generating queries with cardinality constraints for DBMS testing. *IEEE Transactions on Knowledge and Data Engineering*, 18(12):1721–1725, 2006.

[10] L. Cabibbo and A. Carosi. Managing inheritance hierarchies in Object/Relational mapping tools. In *Proceedings of the 17th International Conference on Advanced Information Systems Engineering (CAiSE'05)*, pages 135–150, 2005.

[11] C. Cadar, P. Pietzuch, and A. L. Wolf. Multiplicity computing: a vision of software engineering for next-generation computing platform applications. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, FoSER '10, pages 81–86, New York, NY, USA, 2010. ACM.

[12] A. Cunha and H. Pacheco. Mapping between alloy specifications and database implementations. In *Proceedings of the Seventh International Conference on Software Engineering and Formal Methods (SEFM'09)*, pages 285–294, 2009.

[13] B. Demuth, H. Hussmann, and S. Loecher. OCL as a specification language for business rules in database applications. In *Proceedings of the UML 2001–The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, pages 104–117, 2001.

[14] M. L. Drago, C. Ghezzi, and R. Mirandola. A quality driven extension to the QVT-relations transformation language. *Computer Science - Research and Development*, pages 1–20, 2011.

[15] Ethan K. Jackson, Eunsuk Kang, Markus Dahlweid, Dirk Seifert, and Thomas Santen. Components, platforms and possibilities: Towards generic automation for MDA. In *Proceedings of International Conference on Embedded Software*, pages 39–48, 2010.

[16] Harith T. Al-Jumaily, Dolores Cuadra, and Paloma Martínez. OCL2Trigger: deriving active mechanisms for relational databases using model-driven architecture. *Journal of Systems and Software*, 18(12):2299–2314, 2008.

[17] F. Heidenreich, C. Wende, and B. Demuth. A framework for generating query language code from OCL invariants. *Electronic Communications of the EASST*, 9, Nov. 2007.

[18] S. Holder, J. Buchan, and S. G. MacDonell. Towards a metrics suite for Object-Relational mappings. *Model-Based Software and Data Integration*, CCIC 8:43–54, 2008.

[19] C. Ireland, D. Bowers, M. Newton, and K. Waugh. Understanding object-relational mapping: A framework based approach. *International Journal on Advances in software*, 2:202–216, 2009.

[20] D. Jackson. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(2):256–290, 2002.

[21] W. Keller. Mapping objects to tables - a pattern language. In *Proc. of the European Pattern Languages of Programming Conference*, page 207, 1997.

[22] S. A. Khalek, B. Elkarablieh, Y. O. Laleye, and S. Khurshid. Query-aware test generation using a relational constraint solver. In *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pages 238–247. IEEE Computer Society, 2008.

[23] S. Krishnamurthi, K. Fisler, D. J. Dougherty, and D. Yoo. Alchemy: transmuting base alloy specifications into implementations. In *Proceedings of FSE'08*, pages 158–169, 2008.

[24] S. Q. Lau. *Domain Analysis of E-Commerce Systems Using Feature-Based Model Templates*. Master's thesis, University of Waterloo, Canada, 2006.

[25] S. Philippi. Model driven generation and testing of object-relational mappings. *Journal of Systems and Software*, 77(2):193–207, 2005.

[26] C. d. l. Riva, M. J. Suï£¡rez-Cabal, and J. Tuya. Constraint-based test database generation for SQL queries. In *Proceedings of the 5th Workshop on Automation of Software Test*, pages 67–74, Cape Town, South Africa, 2010. ACM.

[27] S. E. Stemler. A comparison of consensus, consistency, and measurement approaches to estimating interrater reliability. *Practical Assessment, Research and Evaluation*, 9(4), 2004.