Checking the Correctness of What-If Scenarios

Mariana Carvalho[1], Nuno Macedo[2], and Orlando Belo[1]

[1]ALGORITMI R&D Center
[2]HASLab R&D Center
Department of Informatics, School of Engineering, University of Minho
4710-057 Braga, Portugal

Abstract. What-If analysis tools have been increasingly used over the last years, helping creating hypothetical scenarios using simulation and without harming business activities. By integrating OLAP usage preferences, the What-If process is improved. The main advantage is to provide the user with a way to overcome the difficulties that arise when dealing with the conventional What-If analysis scenario process. This paper describes the formal model of the hybridization methodology that combines What-If analysis and OLAP usage preferences and aims to suggest to the user enriched What-If scenarios based on the usage preferences of a certain user. We specify the hybridization model using Alloy and the Alloy Analyzer, which enables to analyze our hybrid model and allows the verification of the model through graphic forms. We verify the correctness of our hybridization model by checking properties and show how the Alloy Analyzer successfully discovers an inconsistency.

Keywords: Decision Support Systems, What-if Analysis, On-Line Analytical Processing, Usage Preferences, Analysis Systems, Formal Specifications, Alloy.

1 Introduction

The presence of analytical information systems and the availability of techniques and models for exploration and analysis of multidimensional data are no longer a novelty in business environments. In addition, there has been a noticeable increase in the number and quality of data retrieving and handling processes created, developed or used by companies. What-If analysis [6] is one way of gaining some competitive advantage from a better use of information and knowledge. What-If analysis processes allow creating simulation models aiming to explore the behavior of complex business systems caused by change of business variables. What-If process helps analyzing the effects of the variables alteration without endangering the real business. This process would not be possible with a historical data manual analysis. An Online Analytical Processing (OLAP) [9] cube is the most adequate data structure to support a What-If simulation due to its characteristics [6]. But when dealing with complex queries, it is essential to filter the retrieved huge volume of information in a way that data do not lose significance, being adjusted according to the users' needs [7] [12].

One of the pitfalls of a conventional What-If analysis is the lack of expertise of the user. Often the user is not familiar with the data or simply may not choose the most correct parameters in the scenario, providing an outcome that probably is not adequate. Therefore, we developed a hybridization methodology [3], which consists in the integration of the OLAP usage preferences in the What-If scenarios process. The hybridization process suggests the user scenario parameters that are strongly related to the goal analysis, introducing valuable information in the conventional what-if analysis application scenario. The hybridization model provides a way to the user to overcome the difficulties that arise when dealing with conventional What-If scenarios process. It is imperative to use formal methods to verify if the hybridization model process meets its critical requirements and provides the desired functionality. With the formal verification, we can create an abstract representation of the hybridization model process in order to detect possible inconsistencies. This paper demonstrates how the formal specification and verification of the hybridization model is made using Alloy [1].

Formal methods [4] are mathematically-based techniques that provide environments and tools, which enable users to specify, verify and analyze models. Formal methods reveal ambiguities, incompleteness and inconsistencies in a system. Formal specifications are not the system implementation, but they describe what a system should do, not how the system should do it. Given a formal specification, it is possible to use techniques to demonstrate that a system design is correct according to its specification. Alloy is a formal object-oriented modeling language based on first-order logic, which gives a mathematical notation for specifying objects and their relationships. Alloy allows creating models that can be automatically checked for correctness using the Alloy Analyzer.

This paper is organized as follows. In section 2 we present an overview about What-If analysis and its applications in formal methods and decision support related areas. Next, in section 3, we briefly introduce formal methods, specification and validation using Alloy. In section 4 we describe the formal specification and verification of the hybridization model in Alloy. Finally, section 5 concludes the paper and discusses some possible future research directions.

2 Related Work

What-if analysis has been increasingly used in several areas, mostly to improve the performance of tools. Klauck et al. [15] presents a simulator tool to interactively model and run generic What-If analysis to help measuring the success of companies, using key performance indicators. De Maio et al. [5] combines the Structural Equation Modeling (SEM) results with Fuzzy Cognitive Map (FCM) to support What-If analysis and get better results. More recently, Meurice et al. [17] presents a tool-supported What-If based approach to detect and avoid inconsistencies in database schemas. Jiang et al. [13] presents a methodology that can estimate the distribution of cloud latency of a request under a What-If scenario. McGarvey et al. [16] shows how to improve an optimization model and decision support tool already developed, including adding What-If capability in order to help MoDOT to identify the impact of

changing policies. Rome et al. [18] presents a system that enables exploring and comparing different courses of action and their consequences using what-if analysis in complex simulated scenarios based on federated modelling, simulation and analysis of Critical Infrastructures. Hung et al. [10] proposes techniques to recommend data ranges for what-if analysis, to find an optimal solution for it, establishing performance guarantees in terms of runtime and result quality.

Only a few papers have focused on both What-If analysis process and formal methods. Most papers intend to study and prevent systems security breaches, which is one of the main applications of formal methods in software systems. Rungta, et al. [19] designs and implements a What-If analysis methodology using formal methods in order to analyze the impact of failures and changes in heterogeneous networks. Two years later, Testa et al. [20] proposed a What-If analysis and robustness checking using formal methods, in order to study and minimize unexpected problems in wireless sensor networks. Augusto et al. [2] addressed the problem of the dependability assessment of Wireless Sensor Networks, which have the ability to perform What-if analysis.

3 Specifying Systems in Alloy

Formal specification languages are usually supported by tools that help validate design decisions by formally verifying desired properties. *Alloy* is one such language that follows a lightweight approach by being simple and flexible, with an object-oriented flavor. This allows the creation of abstract models that can be easily refined into detailed models as the confidence in the design increases. The *Alloy Analyzer* provides automatically and quickly feedback to the user through a graphical visualizer. Alloy is useful to both validate and verify specifications, since assertions can be checked for those models, generating counter-example instances for broken properties. In the latter case, we analyze the counter-example and understand why it happened, where it failed and consequently correct the specification model adding restrictions. If checking an assertion generates no counter-example, that property is guaranteed to hold within the specified universe.

Alloy models are comprised by signatures and relations between their elements. *Signatures* (`sig`) represent classes of entities of the specification, called atoms in Alloy. Relations between atoms are declared as *fields* inside signature declarations. Additional constraints can then be used to further restrict what are considered valid signature and field assignments. A `fact` consist of one or more constraints and represent the assumptions of the model, that will be forced to hold for every instance. Facts can also be attached directly to signatures, in which case it will apply implicitly to every atom of the signature. Predicates (`pred`) and functions (`fun`) are reusable formulas and expressions, respectively, that can be invoked in other parts of the model. Constraints are defined in a relational logic enhanced with transitive closure operations. For the full language reference see [11].

The Alloy Analyzer can be instructed to analyze the specification through `run` and `check` commands, accompanied by *scopes* that specify the size of the universe that will be analyzed. The former instructs the Analyzer to return an example of an

instance consistent with the defined specifications (and additional ones if desired). The latter will instead check whether an assertion (`assert`) holds for the specification. Scopes are defined by specifying the maximum number of atoms allowed for each signature (or the exact number through `exactly`). The design process starts with loose assumptions and is validated through run commands. Once such instances are validated, one starts to define assertions that are expected to hold; the assumptions are refined until no counter-example for the assertions are found.

4 What-If Correctness Verification

4.1 Process Overview

A simulation model is the focus of the What-If process. This model is a simplification of the business model, divide into several analysis scenarios. In turn, each scenario is comprised of a set of variables, related to the business domain (source variables), and by a set of scenario parameters, a group of variables technically related to the simulation. What-if analysis calculates the impact of the changes of the variables in that scenario, presenting the user a new scenario, called the prediction scenario. It is the responsibility of the user to edit the variables and obtain the chosen forecast. Our work is centered on the integration of OLAP usage preferences into the What-If analysis process [3]. This makes it possible to simulate the behavior of a system based on past data extracted from OLAP sessions. Preferences recommend axes of analysis that are strongly related to each other, introducing valuable information in the application scenario. To do that, we designed a specific process for enhancing the What-If application and we need to specify and validate this methodology, especially the generation of OLAP usage preferences, in order to guarantee the effectiveness of this process. We start the process using an OLAP data cube as input, and we define an application scenario based on historical data extracted from previous OLAP sessions. Then, we retrieve the set of preferences based on the association rules retrieved by the OLAP mining process and suggest them to the user as scenario settings. The user is responsible for choosing the scenario settings, which are the axis of analysis, the set of values for analyzing, and the set of values to change according to the user's needs. Afterwards, the What-If process returns an OLAP data cube for prediction support.

Using preferences based on association rules has the advantage that the user does not need to know the business domain. Consequently, in our process, we get more focused and refined results, which help both a user who is not familiar with the business analysis and an analyst who is familiar with the business modeling data.

4.2 Specifying OLAP Usage Preferences

We start by specifying the Data Warehouse [14], describing elements like fact tables, dimensions, measures and attributes, according to the relation' properties between each other. Tables (abstract signature `Table`) can either be fact tables (`FactTable`) or

dimensions (`Dimension`), and the assigned fields (`flds`) represent their records (`Field`, each one forced to belong to exactly one table by the signature constraint). A fact table receives the numerical performance measurements of the business. They are composed by primary keys (usually a set of foreign keys that are related to dimensions) and numeric values, here represented by signature `Measure`. Each row in a fact table corresponds to a measurement event and every foreign key in the fact table has a match to a unique primary key in the respective dimension. Dimensions contain the textual descriptors of the business, here represented by signature `Attribute`. We specify that a fact table can be related (`rels`) with the other existing dimensions (but not with itself).

```
abstract sig Table {
   rels: set Table, flds: some Field
} { this not in rels }
sig FactTable extends Table {} { flds in Measure }
sig Dimension extends Table {} { flds in Attribute }
abstract sig Field {} { one this.~flds }
sig Measure, Attribute extends Field {}
```

The OLAP cube (`OLAPCube`) is constructed using the elements of the data warehouse. The OLAP cube is defined by `fields` that are either measures or attributes. The cube construction is specified using the predicate `ConstructCube`, that constructs an `OLAPCube` provided a representation of user parameters for its creation (`CubeParams`), which are the tables selected to generate the cube. The fields of every selected tables are assigned to the fields of the cube.

```
one sig CubeParams { tabs: set Table }
one sig OLAPCube { fields: set Field }
pred ConstructCube[c: OLAPCube, p: CubeParams] {
c.field = p.tabs.flds }
```

Next, the rules are extracted from the OLAP cube, through OLAP mining [8]. We define the mining structure and the mining model to support a mining association process that runs over the cube and retrieves the rules. The `MiningStructure` defines the data from which mining models are built and the resulting `MiningModel` (`mdl`), created by applying an association rules algorithm to data. This model consists of a set of `rules`, each denoting a logical implication, a rule X –> Y meaning that if X occurs, then it is likely that Y also occurs. The antecedent can be one or more fields (`is`) and the consequent represent a single one (`o`). Each rule is related to a pair of (positive) performance measures (support `supp` and confidence `conf`), which help to identify which rules are relevant. To guarantee that every valid rule is created, signature `SubsetField` is defined to represent the powerset of all available fields.

```
one sig MiningStructure { cube: one OLAPCube, mdl: one MiningModel }
one sig MiningModel {
   rules: set Rule, strongRules: set rules, prefs: set Preference }
sig Rule {
   is: some Field, o: one Field, performance: one Performance }
sig Performance {
   supp: Int, conf: Int } { gte[conf,0] && gte[supp,0] }
sig SubsetFields {
   elems: set Field
} { all s : SubsetFields | s.@elems = elems => s = this }
```

The predicate `ConstructRules` specifies the creation of the rules given a cube and its parameters by creating all the combination of the rules using the cube's fields of the tables in the parameters. The function `reach` helps finding all the fields that are reachable within the OLAP cube given the tables selected in the parameters.

```
pred ConstructRules[c: OLAPCube, p: CubeParams] {
   ConstructCube[c,p]
   all f: c.fields, s: SubsetFields |
      some s.elems && s.elems in related[f,p] =>
         some r: c.~cube.mdl.rules | r.o = f && r.is = s.elems
   all r: (c.~cube.mdl.rules) {
      some f: c.fields | r.is in related[f,p]
      all r': c.~cube.mdl.rules - r | r.is != r'.is || r.o != r'.o }
   }
fun reach[f: Field, p: CubeParams] : set Field {
   ((f.~flds&p.tabs).*(p.tabs <: rels :> p.tabs)).flds - f }
```

The next phase extracts preferences from the discovered mining rules. Relevant rules must be selected from all the rules and item sets extracted from the OLAP cube through the OLAP mining process. To accomplish this, rules are filtered using performance measures' thresholds and an attribute chosen by the user, and stored in the mining model (`strongRules`). These user parameters are encoded by `PrefParams`.

```
one sig PrefParams {
   conf: one Int, supp: one Int, attr: one Attribute
} { gte[conf,0] && gte[supp,0] }
pred ConstructStrongRules[c: OLAPCube, p: PrefParams] {
   c.~cube.mdl.strongRules = { r : c.~cube.mdl.rules |
      p.attr in (r.is + r.o) && gte[r.performance.supp,p.supp] &&
      gte[r.performance.conf,p.conf] } }
```

Strong rules allow us to acknowledge which attributes are strongly related with the chosen attribute. Preferences are built by merging the set of strong rules' attributes. A `Preference` is characterized by a set of fields (`atts`) and a source strong rule (`srcRule`). Preference generation is threefold (`ConstructPrefs`).

```
sig Preference {
   atts: set Field, srcRule: one Rule }
pred ConstructPrefs[c: OLAPCube] {
   all p: c.~cube.mdl.prefs | p.atts = p.srcRule.(is+o)
   all p1,p2: c.~cube.mdl.prefs | p1.srcRule = p2.srcRule => p1 = p2
   c.~cube.mdl.strongRules = c.~cube.mdl.prefs.srcRule }
```

4.3 Validating the Model

Once the process is specified, it must be validated. The first step is to specify the properties that are expected to hold. For rule creation (`RulesCorrect`), for instance, at least one rule must be created, their fields must belong to the cube and all the elements in a specific rule must be unique. An assertion (`CheckRulesBad`) is defined to test whether the construction of the rules guarantees their correctness. The check command instructs the Alloy Analyzer to check the assertion for a particular scope.

```
pred RulesCorrect[c: OLAPCube] {
   some c.~cube.mdl.rules
   c.~cube.mdl.rules.(is+o) in c.fields
```

```
      all r: c.~cube.mdl.rules | r.o not in r.is }
   assert CheckRulesBad {
      all c: OLAPCube, p: CubeParams |
         ConstructRules[c,p] => RulesCorrect[c] }
   check CheckRulesBad
      for 8 but 4 Table, exactly 4 Field, exactly 16 SubsetFields
```

In this case the Alloy Analyzer finds a counter-example that violates the assertion, because it is possible for the parameters to select tables for which no field is reachable from another, rendering the set of rules empty. Thus, an additional restriction must be imposed on the preference selection, which should also be enforced in the implementation of the process: that the selected tables contain reachable fields (`GoodCubeParams`). Once the assertion is fixed to consider this a pre-condition, no counter-examples are found.

```
   pred GoodCubeParams[p: CubeParams] {
      some f: p.tabs.flds | some reach[f,p] }
   assert CheckRulesGood {
      all c: OLAPCube, p: CubeParams |
         (GoodCubeParams[p] && ConstructRules[c,p]) => RulesCorrect[c]
}
```

The creation of the strong rules must also be validated. Given user preferences, predicate `StrongRulesCorrect` defines correct strong rule creation: there is at least one rule, all strong rules contain the preferred attribute, the strong rules are among the set of regular rules, and the performance measures of each strong rule is above the specified threshold.

```
   pred StrongRulesCorrect[c: OLAPCube, p: PrefParams] {
      some c.~cube.mdl.strongRules
      p.attr in c.~cube.mdl.strongRules.(is+o)
      c.~cube.mdl.strongRules in c.~cube.mdl.rules
      all r: c.~cube.mdl.strongRules.performance |
         gte[r.supp,p.supp] && gte[r.conf,p.conf] }
   assert CheckStrongRulesBad {
      all c: OLAPCube, p1: CubeParams, p2: PrefParams |
         (GoodCubeParams[p1] && ConstructRules[c,p1] &&
          ConstructStrongRules[c,p2]) => StrongRulesCorrect[c,p2] }
```

Checking this assertion with the Analyzer also generates a counter-example, illustrated in Fig. 1. In this instance, we have a `FactTable` and a `Dimension`, with certain measures and attributes assigned, respectively. Cube parameters select only the fact table and the preference parameters select `Attribute1`. No strong rule is created because `Dimension`, which contains `Attribute1`, is not part of the cube, so it could not be obtained through the application of the mining algorithm and consequently to be a preference suggested to the user. This counter-example violates the structure and correct function of our process. In order to fix this issue, restrictions must be added to force the chosen attribute to be part of a dimension that belongs to the OLAP cube being mined.

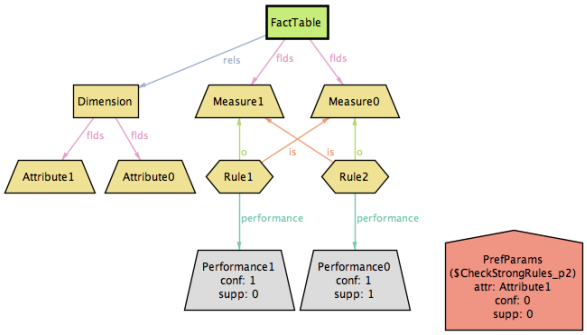

Fig. 1 Counter-example for `StrongRulesCorrect` found by the Alloy Analyzer.

Predicate `GoodPrefParams` holds if the selected attribute belongs to the cube fields and if there is at least one strong rule that passes the given thresholds. Enforcing valid preference parameters, the assertion no longer generates counter-examples, meaning that it is guaranteed to hold for the provided scope. This scope can be increased until the level of confidence in the design is high enough.

```
pred GoodPrefParams[c: OLAPCube, p1: CubeParams, p2: PrefParams] {
   p2.attr in c.fields
   some r: c.~cube.mdl.strongRules.performance |
      gte[r.supp,p2.supp] && gte[r.conf,p2.conf] }
assert CheckStrongRulesGood {
   all c: OLAPCube, p1: CubeParams, p2: PrefParams |
      (GoodCubeParams[p1] && ConstructRules[c,p1] &&
       GoodPrefParams[c,p1,p2] && ConstructStrongRules[c,p2]) =>
         StrongRulesCorrect[c,p2] }
```

It is possible that no instance breaks the assertion due to over-restriction, i.e., by removing suitable instances for the search space. Thus, it is always useful to use run commands to generate valid instances of the specification. Fig. 2 represents one such instance, that was generated with the following command.

```
run {
   some c: OLAPCube, p1: CubeParams, p2: PrefParams |
      GoodCubeParams[p1] && ConstructRules[c,p1] &&
      ConstructStrongRules[c,p2] && GoodPrefParams[c,p1,p2] &&
      ConstructPrefs[c] }
   for 30 but 2 Table, exactly 3 Field, exactly 8 SubsetFields
```

The instance contains a `FactTable` with a measure and a `Dimension` with 3 attributes, both selected by the cube preferences. A set of rules is extracted (`Rule0` through `Rule4`). Taking into account the user `PrefParams` (arbitrarily chosen by the Analyzer) only `Rule4`, from `Attribute1` to `Measure` and confidence 0 and support 6, is considered a strong rule (green bold rule), surpassing the thresholds (confidence 0 and support 6) and containing the selected `Attribute1`. Therefore, the attributes that compose `Rule4` are suggested to the user as a preference (`Preference2`).

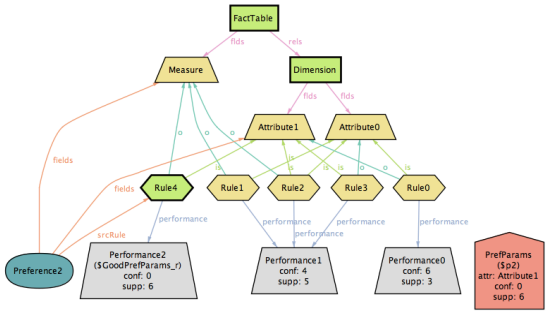

Fig. 2. An example of a consistent instance obtained by the Alloy Analyzer.

5 Conclusions and Future Work

In this paper, we presented and discussed the formal specification and verification of the process of extracting usage preferences in a hybrid model for enhancing What-If scenarios, in order to check for inconsistencies and prove the validity of the model. Alloy was chosen to support such a task. This hybridization model provides us the ability to suggest recommendations, providing the user the most adequate scenario parameters and thus supporting an easier development of a What-If scenario application. OLAP preferences allow for discovering the strongly related axes of analysis based on the usage of a certain user, predicting future demands valuable information of provable new scenarios. We showed how to use Alloy to successfully specify the model, validate some properties of its syntax and also illustrated an example of an instance of our hybrid model running all the Alloy specification. We also presented an example of a counter-example of a situation that come out not following the correct behavior of our syntax model and proposed a solution to correct the assertion that failed. After proposing additional restrictions, the new assertion was checked and no counter-example was found, meaning that our Alloy model is valid and correct within the specified scope. The main advantages of formal specification are to provide a more abstract specification of the process model and for allowing the verification of the model. However, in our work, providing an abstract model is one of the disadvantages, because a formal specification might describe what the system can do, but cannot represent the knowledge extracted and data itself. In our case, the data transformation and recommendations extracted are the focus of our work. As future work, we can validate more properties of our hybrid model and we can investigate further more tools in order to find a way to validate the data itself.

Acknowledgments. This work has been supported by COMPETE: POCI-01-0145-FEDER-007043 and FCT - Fundação para a Ciência e Tecnologia within the Project Scope: UID/CEC/00319/2013 and Project POCI-01-0145-FEDER-016826, which is funded by the European Regional Development Fund (ERDF) through the Operational Programme for Competitiveness and Internationalisation (COMPETE 2020) and by National Funds through the Portuguese funding agency, Fundação para a Ciência e a Tecnologia (FCT).

References

1. Alloy (2017) [Online] Available at: http://alloy.mit.edu/alloy/. [Accessed 24 March 2017].
2. Augusto, J. C., Cinque, M., Coronato, A., Testa, A.: A Formal Methodology to Design and Deploy Dependable Wireless Sensor Networks (2016).
3. Carvalho, M., Belo, O.: Enriching What-If Scenarios with OLAP Usage Preferences. In 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (KDIR'2016), Porto, Portugal, November 9-11, (2016).
4. Clarke, E. M., Wing, J.: Formal methods: State of the art and future directions. ACM Computing Surveys (CSUR), 28(4), pp. 626-643 (1996).
5. De Maio, C., Botti, A., Fenza, G., Loia, V., Tommasetti, A., Troisi, O. Vesci, M.: What-if analysis combining Fuzzy Cognitive Map and Structural Equation Modeling. In 2015 Conf. Technologies and Applications of Artificial Intelligence (TAAI) pp. 89-96 (2015).
6. Golfarelli, M., Rizzi, S. Proli, A.: Designing What-If Analysis: Towards a Methodology. In DOLAP'06, Arlington, Virginia, USA, pp. 51-58 (2006).
7. Golfarelli, M., Rizzi S.: Expressing OLAP preferences. Scientific and Statistical Database Management. Springer Berlin Heidelberg (2009).
8. Han, J.: OLAP mining: An integration of OLAP with data mining. In Proceedings of the 7th IFIP, pp. 1-9 (1997).
9. Harinarayan, V., Rajaraman, A. Ullman, J.: Implementing data cubes efficiently. ACM SIGMOD Record. 25(2) (1996).
10. Hung, N. Q. V., Tam, N. T., Weidlich, M., Thang, D. C., Zhou, X. What-if Analysis with Conflicting Goals: Recommending Data Ranges for Exploration. Proceedings of the VLDB Endowment, 10(5) (2017).
11. Jackson, D.: Software Abstractions: logic, language, and analysis. MIT press (2012)
12. Jerbi, H., Ravat, F., Teste, O., Zurfluh, G.: Preference-based recommendations for OLAP analysis. Springer Berlin Heidelberg (2009).
13. Jiang, Y., Sivalingam, L. R., Nath, S., Govindan, R.: WebPerf: Evaluating What-If Scenarios for Cloud-hosted Web Applications. In Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference. pp. 258-271 (2016).
14. Kimball, R., Ross, M.: The data warehouse toolkit: the complete guide to dimensional modeling. John Wiley & Sons (2011).
15. Klauck, S., Butzmann, L., Müller, S., Faust, M., Schwalb, D., Uflacker, M., Sinzig, W. Plattner H.: Interactive, Flexible, and Generic What-If Analyses Using In-Memory Column Stores. In International Conference on Database Systems for Advanced Applications (pp. 488-497). Springer International Publishing (2015).
16. McGarvey, R. G., Matisziw, T., Noble, J. S., Nemmers, C. J., Karakose, G., Krause, C.: Improving Striping Operations through System Optimization-Phase 2 (2016).
17. Meurice, L., Nagy, C., Cleve, A.: Detecting and Preventing Program Inconsistencies under Database Schema Evolution. In Software Quality, Reliability and Security (QRS), 2016 IEEE International Conference. pp. 262-273 (2016).
18. Rome, E., Doll, T., Rilling, S., Sojeva, B., Voß, N., Xie, J. The Use of What-If Analysis to Improve the Management of Crisis Situations. In Managing the Complexity of Critical Infrastructures (pp. 233-277). Springer International Publishing (2016).
19. Rungta, N., Brat, G., Clancey, W. J., Linde, C., Raimondi, F., Seah, C., Shafto, M.: Aviation safety: model-ing and analyzing complex interactions between humans and automated systems. In Proceedings of the 3rd International Conference on Application and Theory of Automation in Command and Control Systems, pp. 27-37, ACM, May (2013).
20. Testa, A., Cinque, M., Coronato, A., De Pietro, G., Augusto, J. C.: Heuristic strategies for assessing wireless sen-sor network resiliency: an event-based formal approach. Journal of Heuristics, 21(2), pp.145-175 (2015).