Automatic Object Constraint Language Benchmark Generation

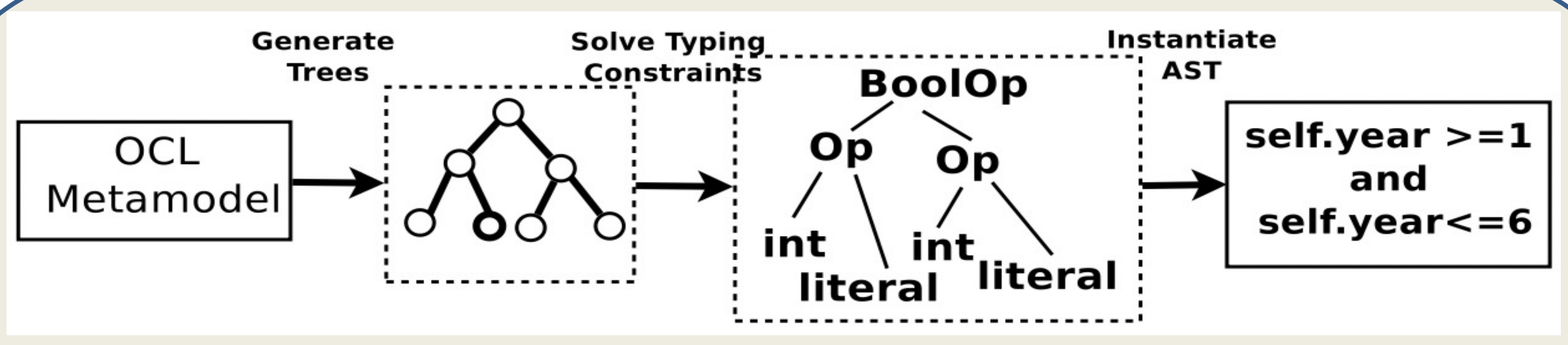
Ankit Jha, Rosemary Monahan, Hao Wu
Principles of Programming Research Group
Department of Computer Science, Maynooth University

1. Overview

Object Constraint Language (OCL), a specification language that allows users to freely express constraints over different model features. One major issue is that it lacks OCL benchmarks which makes difficult to evaluate existing and newly created OCL tools. The main research goals here are

- To develop a domain specific language that allows users to generate customized benchmark.
- To efficiently generate a set of benchmarks.
- To customize benchmarks based on different purposes.
- To evaluate the effectiveness of generated benchmark.

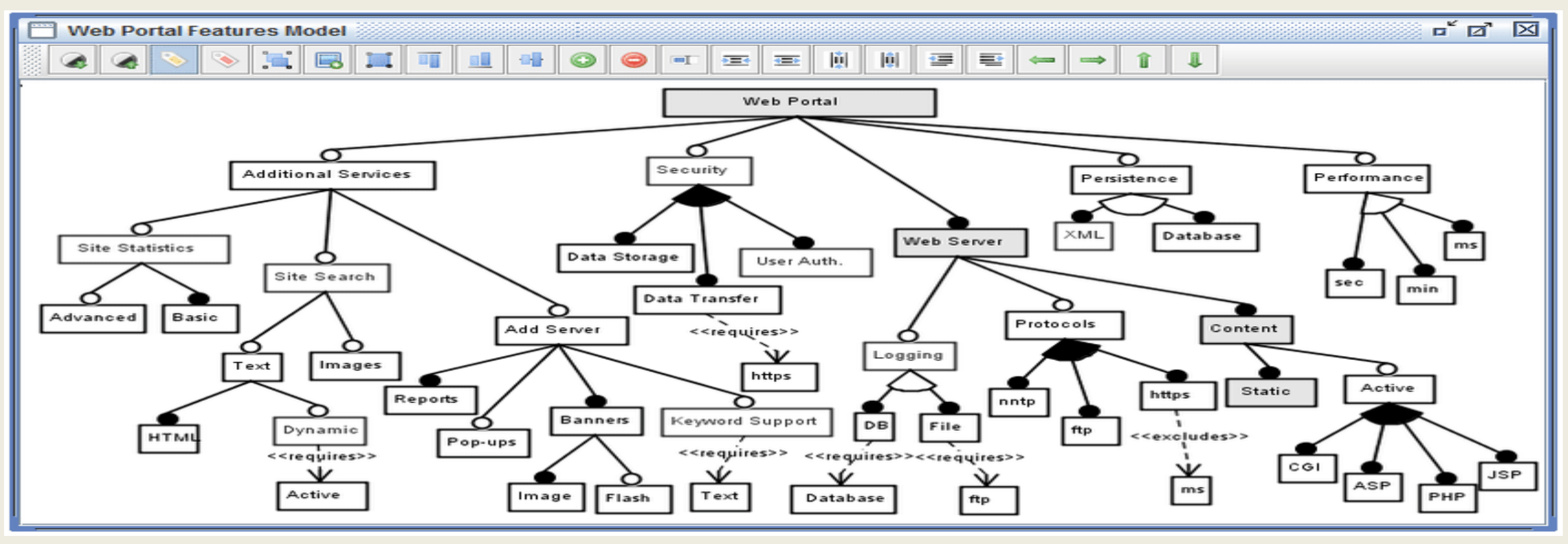
2.The Proposed Idea



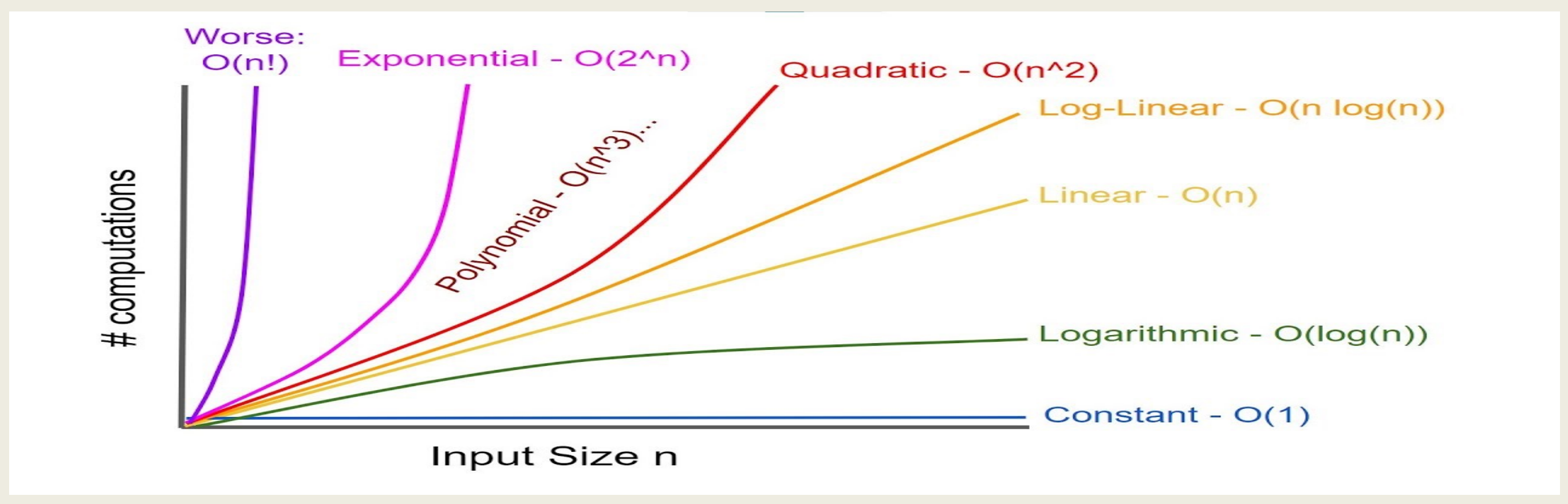
- For a number of OCL constraints to be generated, users first define the properties for each OCL constraint. Here, we consider these properties are described in a standard OCL metamodel.
- Second, we use a tree generator to generate the shape of an abstract syntax tree (AST) for each OCL constraint. This tree generator consults both the OCL metamodel and OCL concrete syntax to produce the ideal size of an AST and generates a set of typing constraints for each AST. These constraints restrict possible types on each node in an AST.
- We then use an SMT solver to solve these constraints to derive a precise type for each node.
- Finally, we traverse the AST and instantiate each node with a concrete value. To form a OCL benchmark, we repeat these steps until the number of OCL constraints a user asked for is met.

4. Challenges & Future Work

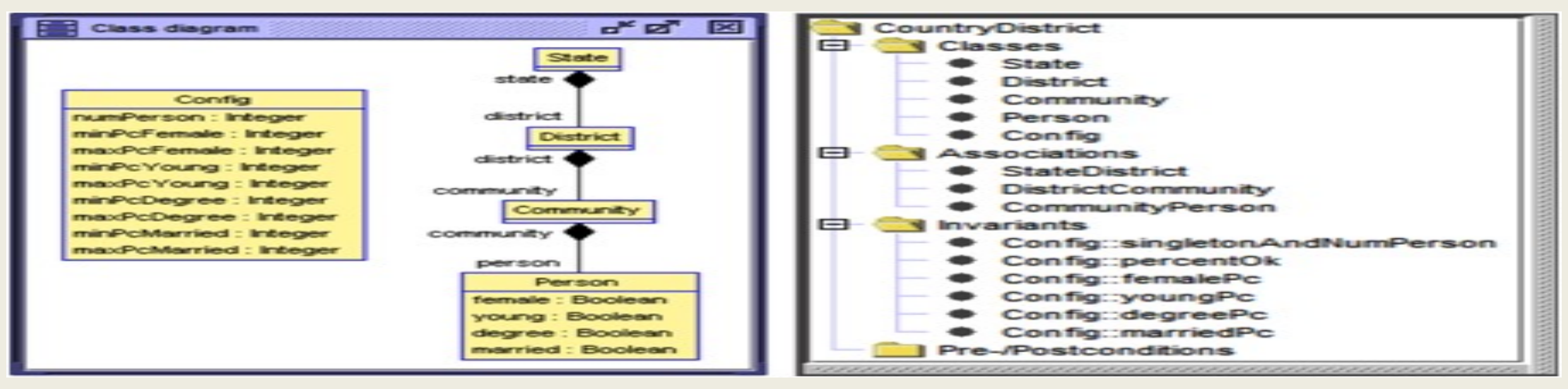
1.Choosing/Designing an appropriate domain-specific language for describing benchmarks. Formally, users would be able to use a well-defined language to describe the kinds of benchmarks to be generated .



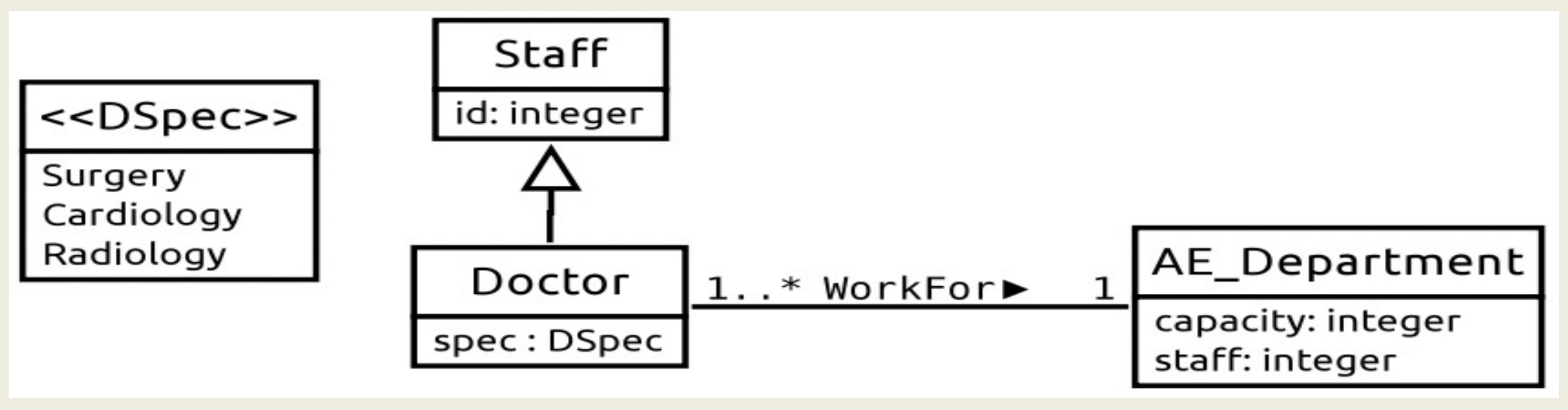
2. Measuring the generated computational complexity of OCL benchmarks using a set of metrics. Users may use different or the same OCL benchmarks for evaluating existing, or their own OCL tools for different purposes.



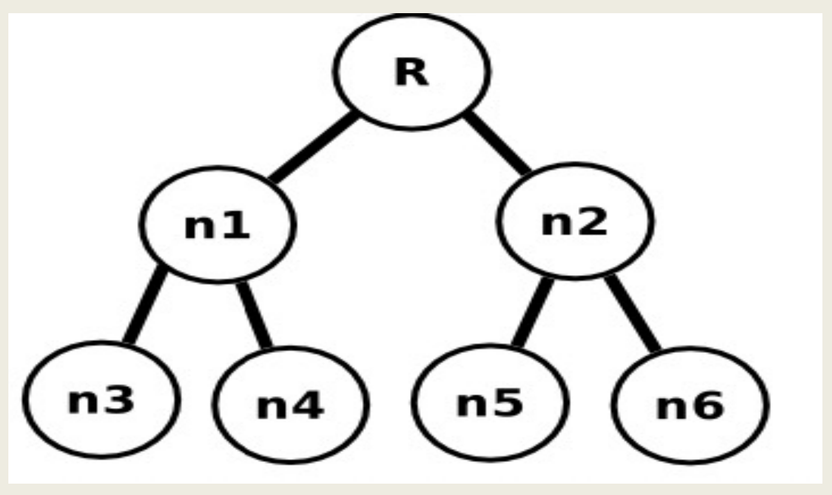
3. Generating OCL benchmarks efficiently and effectively. Typically, the generation process should be completed within a reasonable time frame.



3. An Example



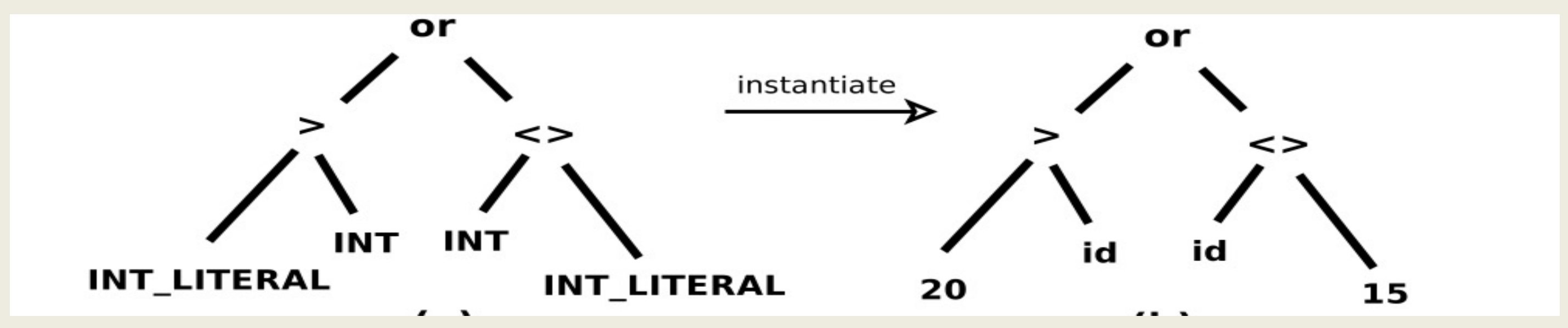
- Consider a scenario where a user has already designed a tool for verifying OCL constraints for given class diagram
- Would like to evaluate the performance and scalability of this tool on the OCL logical expressions with the model.



- We use a tree generator to sketch the shape of an abstract syntax tree based on consulting the OCL concrete syntax .
- A user may select a binary expression for a property constraint over the attribute id. The tree generator then tries to generate a tree that has the specified size.

$$
\begin{aligned}
&(R \in OP_l) \wedge (n1 \in OP_c) \wedge (n2 \in OP_c) \wedge \\
&(T(n3) = INT) \oplus (T(n3) = INT\_LITERAL) \wedge \\
&(T(n4) = INT) \oplus (T(n4) = INT\_LITERAL) \wedge \\
&(T(n5) = INT) \oplus (T(n5) = INT\_LITERAL) \wedge \\
&(T(n6) = INT) \oplus (T(n6) = INT\_LITERAL) \wedge
\end{aligned}
$$

- Since this tree represents a binary expression, the root R must be a binary operator such as > or and. Node n1 and n2 could be another two OCL expressions containing two child node, respectively.
- One of the possible kinds of expressions is that n1 and n2 are two binary expressions as well.
- For the reason of simplicity, let us assume that this is the case. If n1 is a binary expression over id, then either n3 or n4 must be the attribute id1. Similarly, this is the same for n5 and n6.



- To generate constraints for OP_l and OP_c, we use an integer variable to encode each operator and constrain this integer variable to cover all possibilities.
- We use an SMT solver to solve generated typing constraints and interpret the successful assignment for each node in the AST Above figure shows an example of solved type constraints for the AST as mentioned above.
- Finally, we instantiate an AST with concrete values. we use a random value generation for each OCL literal type String, Int and Boolean.
- In this example, we use attribute id for each INT and randomly choose two integers for both INT LITERAL. The final resulting OCL constraint for the attribute id in the class Doctor is shown in above figure

Funded: **Departmental Doctoral Scholarships*(2022-2026)***
Department of Computer Science, Maynooth University