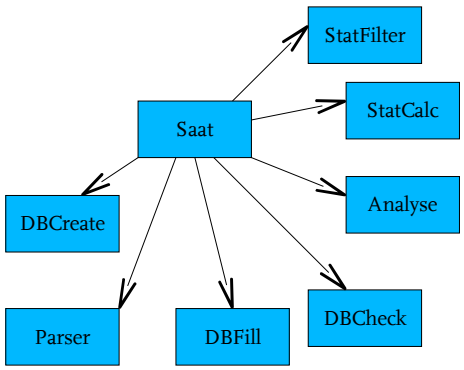
technische universiteit eindhoven

University of Pretoria

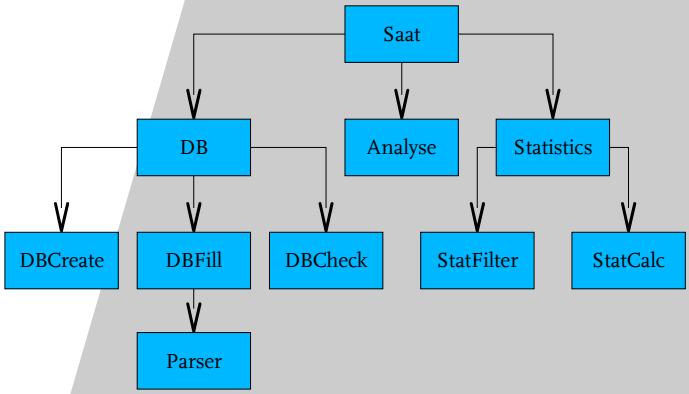
Marc van Kempen (marc@bowtie.nl) Michel Chaudron (m.r.v.chaudron@tue.nl
Derrick Kourie (dkourie@cs.up.ac.za) Andrew Boake (andrew.boake@up.ac.za)

Towards Proving Preservation of Behaviour of Refactoring of UML Models

Refactoring is the transformation of the internal structure of a software system in a way that improves one or more of its quality attributes (such as maintainability, performance), but does not alter the system's external behaviour. Significant gains in development time and effort can be realized if refactoring is used to improve a system's architecture during the design phase. Our goal is to define a collection of refactorings for UML models and develop methods for proving that these refactorings preserve the behaviour of the designs.

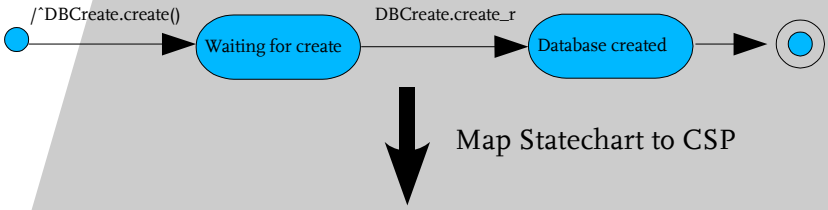


Class diagram before refactoring
Anti-pattern: God-class -> poor maintainability



Class diagram after refactoring
Balanced design: improved maintainability.

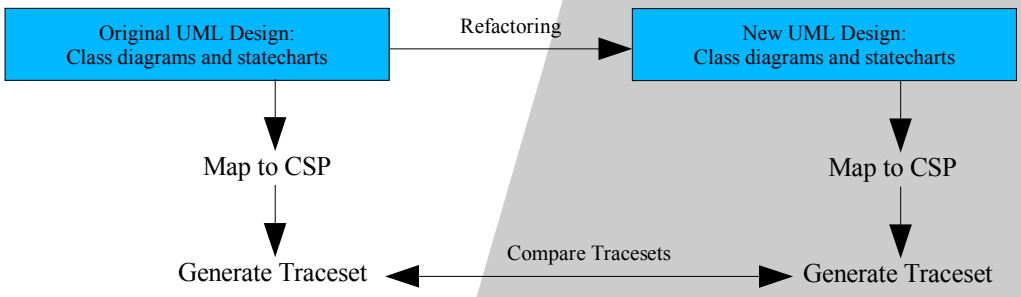
Refactoring a UML design leads to a new set of class and statechart diagrams. We provide a procedure for mapping UML statecharts to the CSP process algebra. We can then use the theory of CSP to prove that the external behaviour of the old and the new design is equivalent.
In the UML model, the behaviour of every class must be described by its own statechart. Further, calling of methods from one class to another is modelled by call events and similarly named return events. A statechart constructed using these constraints can then be mapped to CSP.



Map Statechart to CSP

$$Saat = (cr!create \rightarrow cr_r?x \rightarrow \surd)$$

Mapping example of statechart to CSP.



After refactoring, both statecharts are translated to CSP. Finally the tracesets for both CSP systems are compared to establish equivalence of behaviour.

/department of mathematics and computer science