The Future of Model-Driven Development

Eric James Rapos, Nicolas Chaussé, Juergen Dingel
Modeling & Analysis in Software Engineering Group
School of Computing, Queen's University

Model Driven Development

The process of generating software based on models through some automated form of code generation

Our Group's Work

kiltera

Language for modeling and simulating concurrent, interacting real-time processes; with distribution and mobility

Nicolas' Work

Incremental Test Case Generation

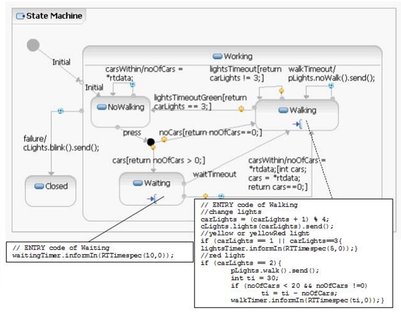
Process of automatic generation of test suites for UML-RT Models, and the effects of model changes on the generated test cases

Eric's Work

Why Modeling Is Important

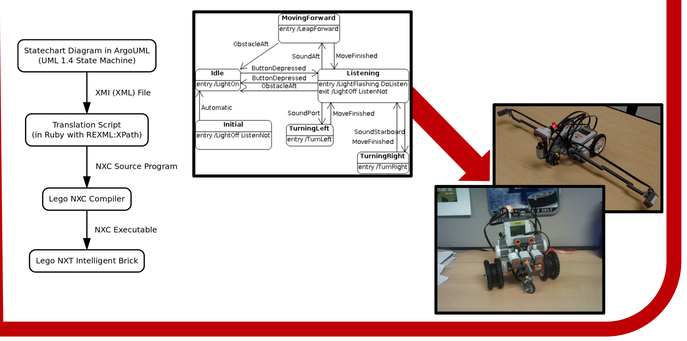
- Software has recently become a very important part of safety critical systems
- Things like missile controls, breaking systems, and others are all controlled by software
- A need exists to ensure the software is correct, but verifiable and provable languages, such as Ada, are too complex
- The solution: MDD through the use of modeling languages/toolkits such as RSA-RTE and kiltera

Rational Software Architect



- The real time edition of RSA (RSA-RTE) is used to create models of systems
- The symbolic execution can then be analyzed to determine correctness

Application: Mindstorms

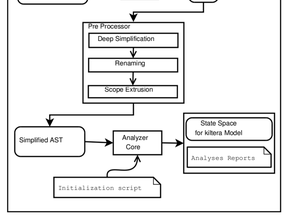


kiltera

Name of the Class	π_{klt} Constructs
PDone	nil
PTrig	x!E
PListener	when{x1?F1@y1 -> P1 |x2?F2@y2 -> P2 ⋮ |xn?Fn@yn -> Pn }
PNew	new x in P
PDelay	wait E -> P
PPar	P1 || P2 || … || Pn
PSeq	P1 ; P2 ; … ; Pn
PDef	def{D1, D2, … Dn}in P
PAssert	assert E -> P
PMatch	match E with { F1 -> P1 | F2 -> P2 … | Fn -> Pn}
PCond	if E then P1 else P2

- Sequential and Parallel Processes
- Channels as first-class objects
- Channels can be passed between processes and PCs
- Channels can be triggered
- Channels can listened to in parallel

Current Analyzer



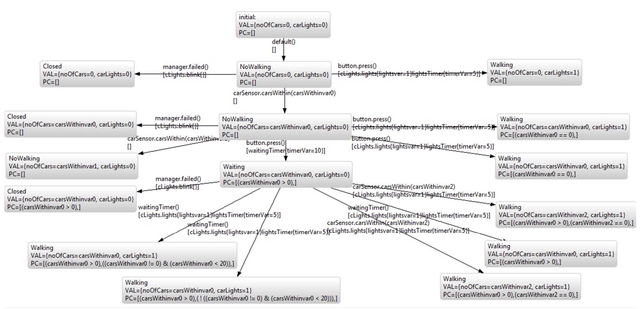
- Uses the Visitor Pattern
- Parses kiltera code and builds the Abstract Syntax Tree
- Builds the State Space Tree
- Produces analyses and reports such as Deadlocks and Stable States
- Will be re-factored using the Language Definition Framework

Refactoring with Language Definition

$$fn(\tau) \stackrel{def}{=} \emptyset \qquad fn(0) \stackrel{def}{=} \emptyset$$
$$fn(x) \stackrel{def}{=} \{x\} \qquad fn(\nu x.P) \stackrel{def}{=} fn(P)\backslash\{x\}$$
$$fn(\bar{x}) \stackrel{def}{=} \{x\} \qquad fn(\alpha.P) \stackrel{def}{=} fn(\alpha) \cup fn(P)$$
$$fn(P_1 + P_2) \stackrel{def}{=} fn(P_1) \cup fn(P_2)$$
$$fn(P_1 \parallel P_2) \stackrel{def}{=} fn(P_1) \cup fn(P_2)$$

CCS is a Process Calculus
CCS ⊂ π-calculus ⊂ π_{kil} ⊂ kiltera

Symbolic Execution

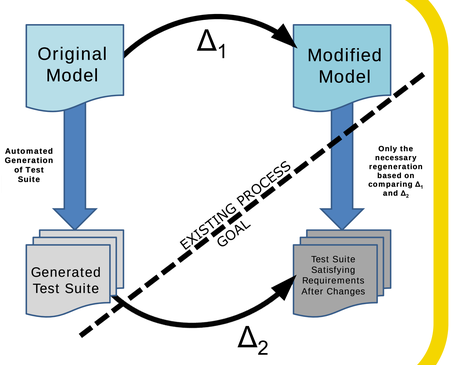


Reasons for Incremental Test Case Generation

- Automated generation of a full test suite can become a costly operation
- By determining and making only the necessary changes, this cost can be reduced
- Incrementally generating the test cases as modifications are made will ensure correctness of the test suite

The Process

- Find relationship between Δ_1 and Δ_2
- Determine which changes need to be made to the test suite



Resources

❖ Engin Uzuncaova, Sarfraz Khurshid, Don S. Batory: Incremental Test Generation for Software Product Lines. IEEE Trans. Software Eng. 36(3): 309-322 (2010)

❖ J. Dingel, E. Paen, E. Posse, R.R. Rahman, and K. Zurowska. Denition and implementation of a semantic mapping for UML-RT using a timed pi-calculus. In Proceedings of the Second International Workshop on Behaviour Modelling: Foundation and Applications, pages 1-8. ACM 2010

❖ R. Rahman. Design and Implementation of an Analyzer for a Timed π -calculus. 2010

❖ E. Posse. Symbolic simulation of π_klt. 2010