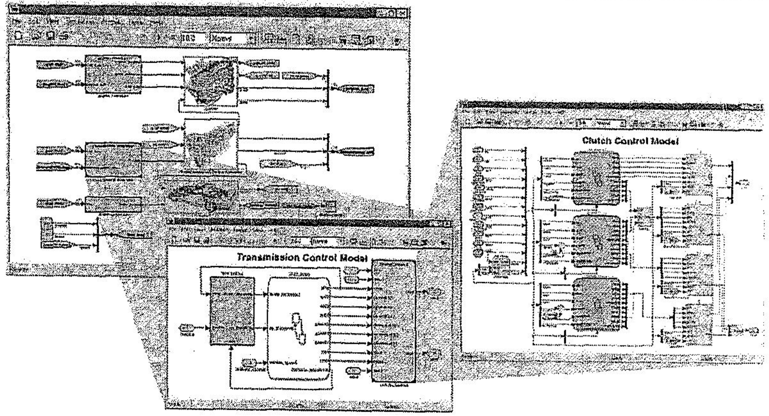
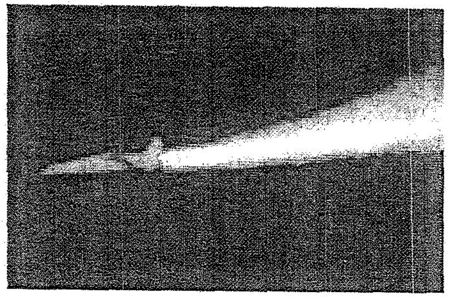
A Software Safety Certification Plug-in for Automated Code Generators

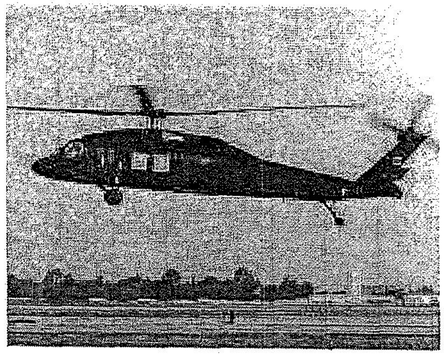
PI: Ewen Denney, USRA/RIACS
Johann Schumann, USRA/RIACS
Doug Greaves, NASA Ames

Auto-generated code at NASA

- c.50% of NASA missions and projects use modeling tools like Simulink and Matlab
- Commercial code generators (e.g., Real-Time Workshop and MatrixX) are available and have been successfully used
  - X-43 Hyper-X: On-board flight-software generated from Simulink models
  - RASCAL: Helicopter control laws implemented using Real-Time Workshop







"We never found any errors in the automatically generated code, so we were confident in creating a quick prototype for NASA." (P. Seigman, Boeing)

Safety of auto-generated code

- “Experience shows everything is fine…”
- A look into RT Workshop shows:

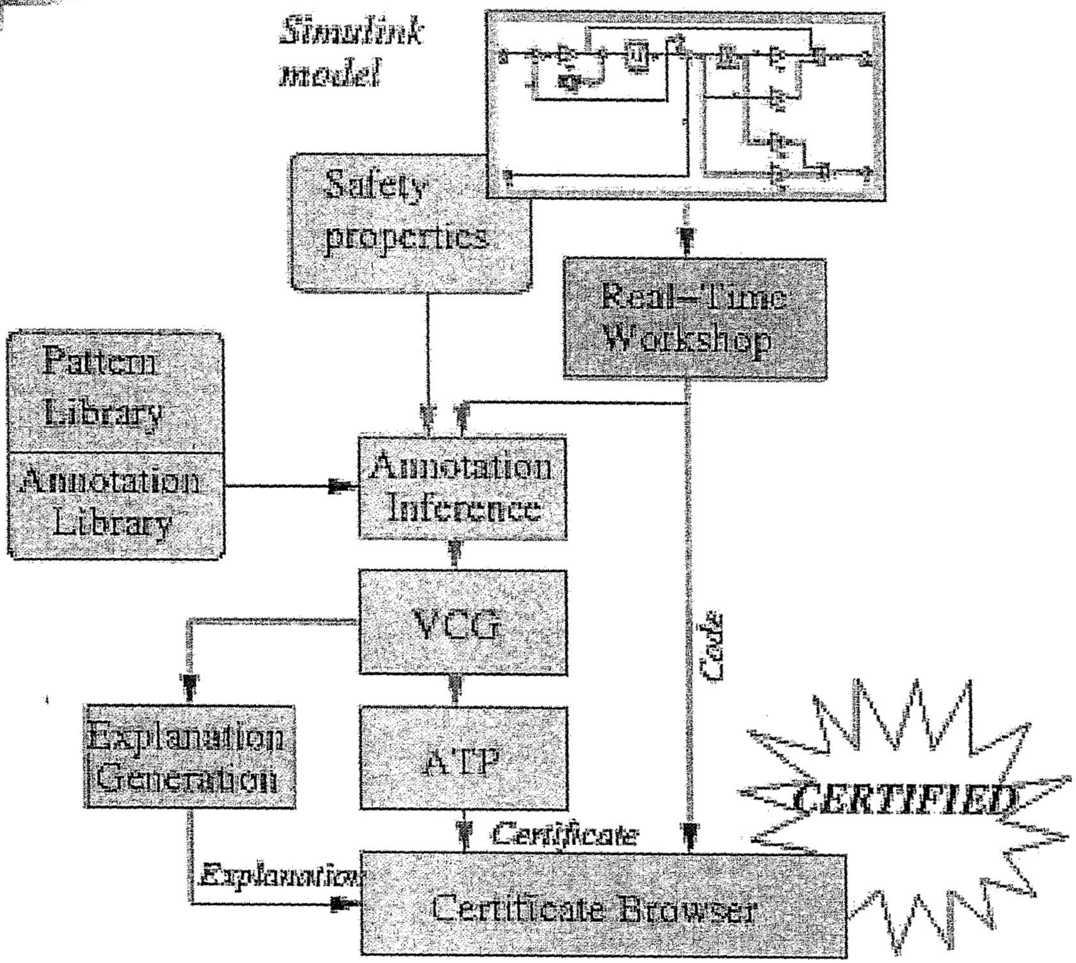
```
/* Copyright 1994-2002 The MathWorks, Inc.
...
for (;;) {
  utAssert( (x[bottom] < u) && (u < x[top]) );
  idx = (bottom + top)/2;
  if (u < x[idx]) {  top = idx - 1;
  } else if (u >= x[idx+1]) {
              bottom = idx + 1;
  } else {    return(idx);
  }}
```

Code used for Simulink table interpolation

Assertion does not match with program. If activated, can lead to program abort

For safety-critical and human-rated applications, good experience is not enough. IV&V needs formal tools to check safety of auto-generated code

Technical approach



- Combine generator with certification tool
- Generate certificates which can be verified independently (IV&V)
- Based on formal logic
  - Hoare-style safety verification
  - Range of safety properties
  - Pattern-based approach to inferring annotations
  - Fully automated
  - Small set of trusted components

Safety properties

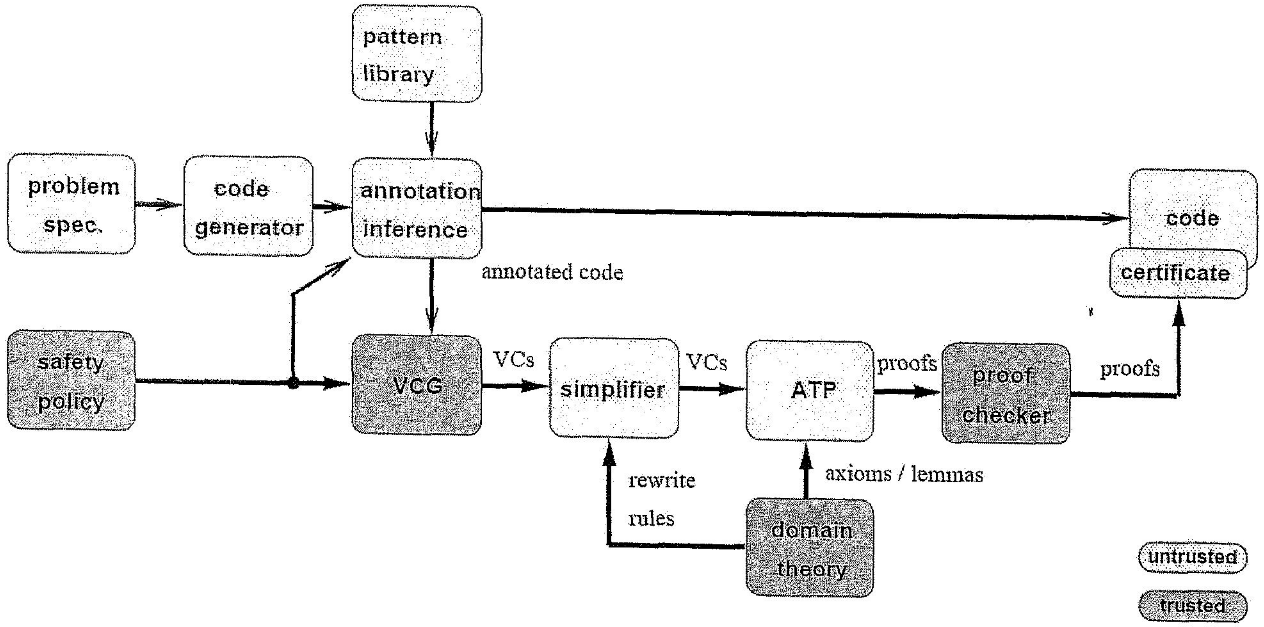
- Language-specific
  - Variable initialization before use, array bounds
- Domain-specific
  - “block properties”
    - “all values of *x* in interpolation table disjoint and increasing”
  - “system-specific properties”
    - Signals used, transients, numerical properties, stability, …
- Project-specific
  - Flight rules …

Annotation Inference

- Formal basis
  - Express safety properties in Hoare logic
  - VCG generates verification conditions from *annotated* code (pre/post-conditions and invariants)
- Patterns
  - Express common coding idioms
  - Generate from core set
  - e.g. for i := _ to _ do x[i, i notin _] := _
- Annotation schemas
  - Add annotations for given safety policy
- E.g. Matrix init -> annotated init

Trusted components

- Small kernel of trusted components
- Avoids costly tool qualification
- Could be used to build safety case
- Obtain verification credit



Safety explanations

- Verification says that the code is safe
- Explanation explains *why* it's safe

  ⇒ Safety documentation

  ⇒ Code reviews
- Explanation mechanism:
  - Extend logical rules with mark-up
  - Turn marked-up VCs into text

Certification browser

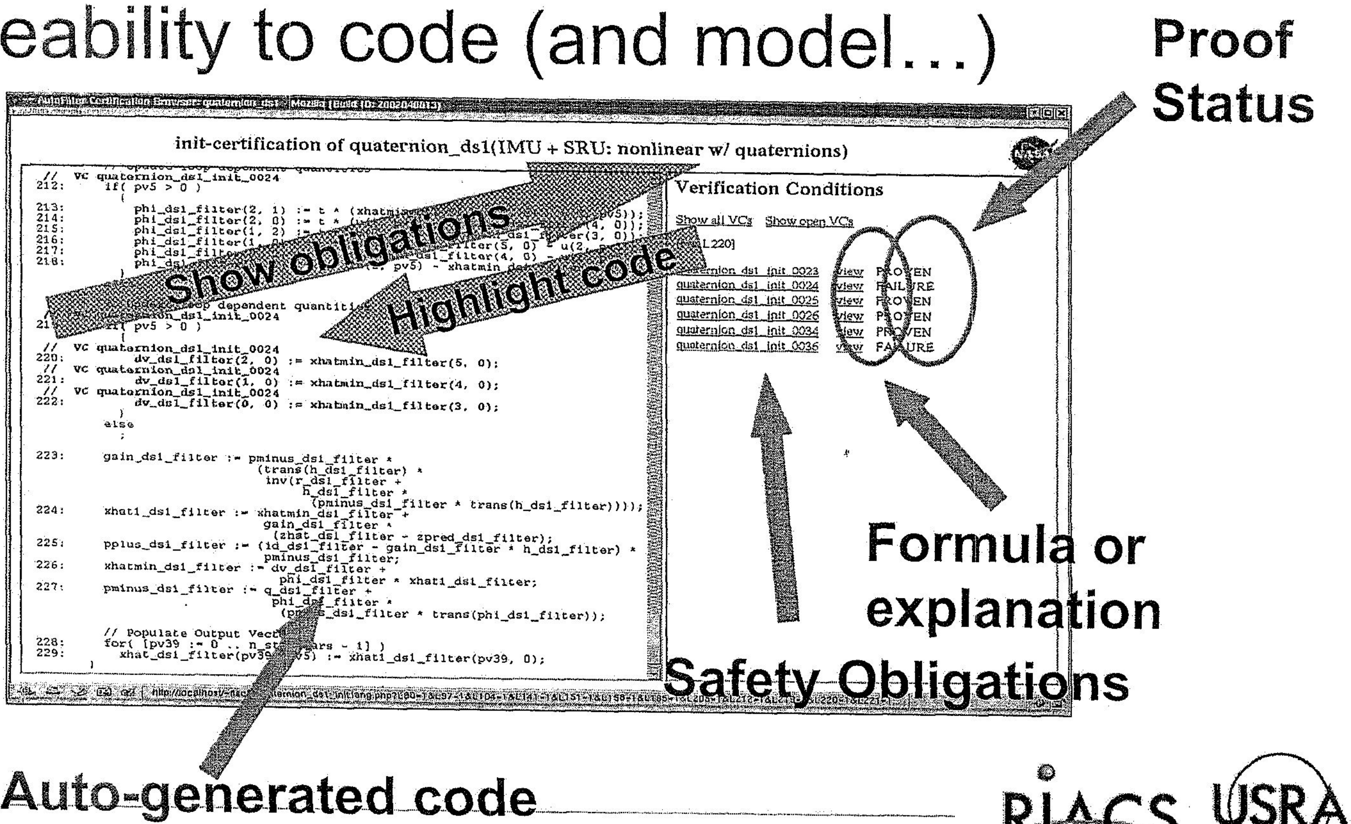
- Integrate prototype of certification browser with RTW using Matlab's guide interface builder
- Depending on traceability information provided by RTW: develop basic traceability functionality from VCs to Simulink blocks
- Defer safety explanations to Phase II

Audit support

- Display and explanation of proof tasks
  - explain *why* code is safe

    "the table lookup, f[b+i], at line 23, is within bounds because ..

    => support code reviews
- Traceability to code (and model…)



AutoFilter Certification Browser: quaternion_ds1 - Mozilla {Build ID: 2002040813}

init-certification of quaternion_ds1(IMU + SRU: nonlinear w/ quaternions)

```
 //  VC quaternion_ds1_init_0024
212:      if( pv5 > 0 )
            {
213:          phi_ds1_filter(2, 1) := t * (xhatmi                    pv5));
214:          phi_ds1_filter(2, 0) := t *                        (4, 0));
215:          phi_ds1_filter(1, 2)                     ds1_filter(3, 0));
216:          phi_ds1_filter(1,                     filter(5, 0) - u(2,
217:          phi_ds1_filt                       ds1_filter(4, 0) -
218:          phi_ds                            pv5) - xhatmin
            }

              loop dependent quantities
              on_ds1_init_0024
21        if( pv5 > 0 )
            {
 //  VC quaternion_ds1_init_0024
220:          dv_ds1_filter(2, 0) := xhatmin_ds1_filter(5, 0);
 //  VC quaternion_ds1_init_0024
221:          dv_ds1_filter(1, 0) := xhatmin_ds1_filter(4, 0);
 //  VC quaternion_ds1_init_0024
222:          dv_ds1_filter(0, 0) := xhatmin_ds1_filter(3, 0);
            }
          else
            ;

223:      gain_ds1_filter := pminus_ds1_filter *
                              (trans(h_ds1_filter) *
                               inv(r_ds1_filter +
                                   h_ds1_filter *
                                    (pminus_ds1_filter * trans(h_ds1_filter))));
224:      xhat1_ds1_filter := xhatmin_ds1_filter +
                               gain_ds1_filter *
                                (zhat_ds1_filter - zpred_ds1_filter);
225:      pplus_ds1_filter := (id_ds1_filter - gain_ds1_filter * h_ds1_filter) *
                               pminus_ds1_filter;
226:      xhatmin_ds1_filter := dv_ds1_filter +
                                 phi_ds1_filter * xhat1_ds1_filter;
227:      pminus_ds1_filter := q_ds1_filter +
                                phi_ds1_filter *
                                 (pm s_ds1_filter * trans(phi_ds1_filter));

          // Populate Output Vect
228:      for( [pv39 := 0 .. n_st    ars - 1] )
229:        xhat_ds1_filter(pv39,  v5) := xhat1_ds1_filter(pv39, 0);
        }
```

Verification Conditions

Show all VCs   Show open VCs

[... L220]

quaternion_ds1_init_0023	view	PROVEN
quaternion_ds1_init_0024	view	FAILURE
quaternion_ds1_init_0025	view	PROVEN
quaternion_ds1_init_0026	view	PROVEN
quaternion_ds1_init_0034	view	PROVEN
quaternion_ds1_init_0036	view	FAILURE

http://localhost/~fisch ternion_ds1_init/ang.php?L80=1&L97=1&L104=1&L141=1&L151=1&L158=1&L185=1&L2

Show obligations

Highlight code

Proof Status

Formula or explanation

Safety Obligations

Auto-generated code

Goal at 6 months

- Demonstrate fully automated verification on useful subset of Simulink blocks for limited range of policies on selected examples
- PDR
- “feasibility study and report”
  - How we adapted technology
  - How we applied it to examples
  - Report on PDR
  - Why it’s a good thing: technically and practically
  - Plan for rest of project

Extra Slides

Project plan

- Phase I (6 months): c. start June – end November
  - Determine how certification machinery must be adapted for Real-Time Workshop
  - Case study: Auto-generated code from Ames (e.g. Vertical Motion Simulator)
- Phase II
  - Extend and mature prototype
    - Y2: more modeling features, patterns, …
    - Y3: domain- and project-specific safety policies (flight rules)
  - Deliverable: Cert/RT – certification tool for RTW
  - Second case study (e.g. NASA Dryden)

Project participants

- Ewen Denney, USRA/RIACS
- Johann Schumann, USRA/RIACS
- Doug Greaves, Code AFJ, NASA Ames
- Bernd Fischer, Uni. Southampton
- Programmer (TBD)

Deliverables

Task #	Deliverable Title	Description/Content	Due Date* (YYYY/MM/DD)	For publication? (Y, N)
T1	D1: Report on study and PDR of Cert/RTW Study	Report on feasibility study of certification technology on selected Simulink models and Code; report on Preliminary Design review	2006/08/30	Y
T2, T3	D2: Report on Cert/RTW Prototype architecture	Report on architecture of Cert/RTW (Critical Design review) and report on initial certification patterns and extension of domain theory	2007/02/28	Y
T4, T5	D3: Cert/RTW Alpha	Report on features and capabilities of Alpha Version	2007/08/30	Y
T6	D4: Cert/RTW Beta	Report on features and capabilities of Beta Version	2008/02/28	Y
T7	D5: Report on Case Study II	Report on Case Study II including evaluation of tool and plan for tool robustification	2008/11/30	Y
T8	D6: Final Report	Final Project Report; Cert/RTW User Manual; Cert/RTW delivered to IV&V	2009/2/28	Y

*Relative to project start date

Case study

- Models as driving examples
  - Simulink diagrams + generated code (+ some explanation ;-))
  - How to generate code (command line parameters, config sets and settings)
  - Required environment (if any), version of RTW
  - Code architecture
- Modeling conventions
  - Subset of Simulink blocks used for RASCAL
  - Modeling standards (cf. MAAB)

Case study

- Software development and V&V process
  - Specific modeling process?
  - Testing, simulation, code reviews
- Properties to check
  - Software specific: array bounds, division-by-0, uninitialized variables
  - Model-specific: signals used, transients, numerical properties
  - Flight rules
- Known bugs, issues with RTW
- Feedback on tool and approach

Technical work

- Extensions to certification architecture
  - VCG (logic, coding constructs)
  - Patterns (RTW idioms)
  - Domain theory, prover (block properties)
- Integration
  - Back end: Parser/translator from RTW output
  - Front end: GUI (Matlab UI builder)

Other approaches

- Manual review
  - Time-consuming, laborious
- Exhaustive testing
  - Combinatorial explosion (*n*-inputs, *m*-outputs)
- Post-hoc formal V&V
  - High rate of false negatives
  - No explicit evidence
  - Can require user interaction