Application of Design for Verification with Concurrency Controllers to Air Traffic Control Software

Aysu Betin-Can, Tevfik Bultan, Mikael Lindvall, Benjamin Lux, Stefan Topp

Automated Software Engineering 2005

Content

Current state of US airspace

- Modern Air Traffic Control (ATC) depends on air traffic controllers to maintain aircraft separation
- Surveils radar data for potential conflicts
- Provides clearances to alter the trajectories
- US airspace operates at only half its potential capacity
- Reason: controllers' workload limits
- Little gain from resectorization or new ATC decision support tools

Current state of US airspace

- Modern Air Traffic Control (ATC) depends on air traffic controllers to maintain aircraft separation
- Surveils radar data for potential conflicts
- Provides clearances to alter the trajectories
- US airspace operates at only half its potential capacity
- Reason: controllers' workload limits
- Little gain from resectorization or new ATC decision support tools

Automated Airspace Concept (AAC)

- Work of Dr. Heinz Erzberger (NASA)

- A new paradigm is required to utilize full airspace capacity
- Automated mechanisms play a primary role in aircraft separation
- A persistent 2-way link with ground-based automated system
- Conflict alerts and air traffic control clearances transmitted as data

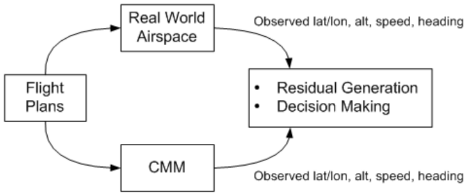
Tactical Separation Assisted Flight Environment (TSAFE)

- Integral part of AAC
- Monitors aircrafts for potential violations of separation
- Computes and issues short conflict-free trajectories otherwise
- Originally implemented by Gregory G. Dennis (MIT) in 2003 as a Master Thesis project
- Later integrated into an experimental environment as part of NASA's High Dependability Computing Project

TSAFE operation

1. A flight is compared to its up-to-date flight plan
2. It is assigned a one of 64 conformance statuses (8h x 8v)
3. Two trajectories are computed: *nominal* and *dead reckoning*
4. Depending on the conformance status one or both are probed for loss of legal separation with other flights
5. If a high risk conflict warning is issued a short term conflict-free trajectory is computed

Calculating conformance


Real World Airspace
Observed lat/lon, alt, speed, heading
Flight Plans
• Residual Generation
• Decision Making
CMM
Observed lat/lon, alt, speed, heading

Conformance monitoring model (CMM) employed to obtain the expected behavior:

- Altitude and speed taken directly from flight plan
- Expected heading and position derived from the nearest point on flight plan to the aircraft's actual position (snap-back)

Calculating trajectories

Dead reckoning trajectory:

- a straight line with current heading and speed

Planned trajectory:

- route from snap-back point to the next fix
- if look ahead time is greater than the time to reach the fix, add route to the next fix
- iterate until unable to reach fix in the set time horizon

Demonstration

Demonstration of TSAFE

Design for verification

Verification of complex systems depends on extracting a simplified model

- Requires reverse engineering to rediscover many properties known to developers at design time

In *design for verification* developers either document their decisions or apply specific agreed-upon design patterns.

TSAFE III application character

TSAFE III

- Java client/server application
- server performs the conformance monitoring and trajectory synthesis
- client GUI in Swing/AWT
- parallel threads access a shared database
- RMI calls between client and server threads
- 21k lines of code in 87 classes

Java concurrent programming is error prone (conditional waits/notifications using synchronized, wait, notify, notifyAll)

TSAFE verification

Application has been reengineered to utilize concurrency controllers

- concurrency controller interface verified separately from threads
- threads participate in concurrency only through outside interfaces

Verification can be split to verifying the controller interfaces and that threads obey the contracts of those interfaces.

Concurrency controllers

- Multiple client threads can read data from the shared database
- Several server threads can update the data
- RW controller class policies the reader/writer synchronization
- Mutex controller policies exclusive access synchronization

```
class RWController implements RWInterface {
 int nR; boolean busy;
 w_enter = new GuardedCommand() {
  public boolean guard() { return (nR == 0 && !busy);}
  public void update() { busy = true } }; ...
```

Translated to Action Language:

```
w_enter: pc=IDLE and nR=0 and !busy and busy='true and pc='WRITING
```

Verifying controllers

Controllers were verified using the Action Language Verifier (ALV) against a list of properties.

$AG(busy \Rightarrow nR = 0)$

$AG(busy \Rightarrow AF(\neg busy))$

$AG(\neg busy \wedge nR = 0 \Rightarrow AF(busy \vee nR > 0))$

$\forall x AG(nR = x \wedge nR > 0 \Rightarrow AF(nR \neq x))$

...

Interface verification

- Using Java Path Finder (JPF)
- Adhesion to the controller interfaces
- Access the shared data only at allowed interface states
- Each thread type is verified separately - interaction through stubs and from drivers

Stub:

- Concurrency controller - finite state automaton
- Shared data - assertions based on associated concurrency controllers, return any valid data or exception
- RMI, GUI and I/O operations return any exception or value

Interface verification (continued)

Drivers:

- Simulate thread creation by assigning command-line arguments and initialized variables
- Model implicitly created threads - GUI and RMI event threads generating event sequences

Data dependency analysis:

- Verifying a thread with regard to all possible input causes JPF to run out of memory
- Some input parameters or return values don't affect synchronization behavior
- Dependency graphs are generated through bachward traversals
- Values that don't influence the synchronization are set constant
- Finite domain values are enumerated
- Other are provided by the user

Interface verification (continued)

Drivers:

- Simulate thread creation by assigning command-line arguments and initialized variables
- Model implicitly created threads - GUI and RMI event threads generating event sequences

Data dependency analysis:

- Verifying a thread with regard to all possible input causes JPF to run out of memory
- Some input parameters or return values don't affect synchronization behavior
- Dependency graphs are generated through bachward traversals
- Values that don't influence the synchronization are set constant
- Finite domain values are enumerated
- Other are provided by the user

Experiments

Fault seeding:

- 40 modified versions were created
- Benchmark of the verification process
- All concurrency controller faults were discovered
- Not all interface faults were discovered

Missed faults:

- Deep faults - branching dependent on a counter set to 1000, 10000, 100000 passes

Experiments

Fault seeding:

- 40 modified versions were created
- Benchmark of the verification process
- All concurrency controller faults were discovered
- Not all interface faults were discovered

Missed faults:

- Deep faults - branching dependent on a counter set to 1000, 10000, 100000 passes

The End

Thank you for your attention

Discussion