Assured Reconfiguration of Embedded Real-Time Software

By E. Strunk and J. Knight

Claims

? They present a comprehensive approach to assured reconfiguration
? Provide a framework for formal verification that allows system developer to use a set of application-level properties to show general reconfiguration properties
? State their approach allows the system design to be much simpler and easier to analyze
What you should learn from this paper?

- When designing a system, you will have high-level properties/goals you want to satisfy.
- It may be difficult to prove them formally in your design.
- Instead, use low level properties that can be easier to guarantee and use these properties to prove the high-level ones.
- You must be able to formally describe these properties in a language of your choosing.
- In essence, that is what they did here, so this paper is a good example of how to do that.

Definitions

- Specified set $S$ of functionality levels.
- Reconfiguration defines transitions between pairs of members of $S$.
- Informally, the process through which a system halts operation under its current source specification $S_i$ and begins operation under a different target specification $S_j$. 
Terms used

- $S$: the set $\{S_1, S_2, ..., S_n\}$ of service specifications of the system
- $E$: a set of possible functions from factors that affect which specification is an appropriate instantiation of $S_j$ to possible values of those factors
- $\text{Env}(t) \rightarrow e \in E$: function that returns the value of the environmental state at time $t$
- $\text{Choose}(S_x, e \in E) \rightarrow S_y$: function that returns appropriate target elements of $S$, given source elements of $S$ and $E$.

More terms used

- $\text{Pre}_x$: the precondition that must be satisfied for the system to operate under $S_x$
- $\text{Inv}_i$: an invariant that must hold during the transition from $S_i$ to $S_j$
- $\uparrow S_x$: the event marking the beginning of the system’s operation under specification $S_x$. This event occurs when the system first operates according to the function set out in $S_x$, and concurrently satisfies $\text{Pre}_x$.
- $\downarrow S_x$: the event marking the end of the system’s operation under specification $S_x$ (we define the occurrence of a reconfiguration signal to imply $\downarrow S_x$)
- $T_{ij}$: the maximum allowable time between $\downarrow S_i$ and $\uparrow S_j$
Conditions that must hold for reconfiguration $R$

**P1.** $\lceil R, b \rceil = \lfloor S_i, a \rfloor$

(R begins at the same time the system is no longer operating under $S_i$)

Note that this property defines the beginning of $R$ rather than specifying a requirement that should be met.

**P2.** $\lfloor R, b \rfloor = \lceil S_i, c \rceil$

(R ends at the same time the system becomes compliant with $S_i$)

**P3.** $\exists t$: time s.t. $\lceil R, b \rceil \leq t \leq \lfloor R, b \rfloor \land$

$\text{Choose}(S_i, \text{Env}(t)) = S_i \land t >$

($S_i$ is the proper choice for the target specification at some point during $R$)

Continued

**P4.** $\lceil R, b \rceil - \lceil R, b \rceil \leq T_{ij}$

(R takes less than or equal to $T_{ij}$ time units)

**P5.** $\text{Inv}_{ij} < \lceil R, b \rceil, \lfloor R, b \rfloor >$

(The transition invariant holds during $R$)

**P6.** $\text{Pre}_{ij} < \lfloor R, b \rfloor, \lceil R, b \rceil >$

(The precondition for $S_i$ is true at the time $R$ ends)

**P7.** $\lfloor S_i, c-1 \rfloor < \lceil R, b \rceil \Rightarrow \lceil S_i, c \rceil = \lfloor R, b \rfloor$

(The lifetime of $R$ is bounded by any two occurrences of the same specification)
Architecture for Assurance

Claim: If system implementer builds system using this architecture and shows the low-level properties required of the architectural elements, he will know the high-level properties that assure reconfiguration have been met.

2 major components
- Application that performs computations associated with members of S
- Reconfiguration mechanism that ensures reconfiguration can be carried out

Building blocks (events, actions,...)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>App</td>
<td>Action representing operation of the application. Operates continuously throughout R.</td>
</tr>
<tr>
<td>RM</td>
<td>Action representing operation of the reconfiguration mechanism.</td>
</tr>
<tr>
<td>↑RM</td>
<td>Event that App signals fault at top level</td>
</tr>
<tr>
<td>↓RM</td>
<td>$\equiv \downarrow \text{Trans}$ (below)</td>
</tr>
<tr>
<td>↓S_i</td>
<td>Event that a reconfiguration signal is sent to $S_i$</td>
</tr>
<tr>
<td>Halt</td>
<td>Action of App causing $S_i$ to meet Post</td>
</tr>
<tr>
<td>↑Halt</td>
<td>$\equiv \downarrow S_i$</td>
</tr>
<tr>
<td>Post</td>
<td>Predicate that must be true of App in order for reconfiguration to begin. This condition protects data, and ensures that the software is fail-stop.</td>
</tr>
<tr>
<td>Choose</td>
<td>Action of RM determining member of $S$ for $S_j$</td>
</tr>
</tbody>
</table>
More building blocks

\( \text{Pre}_{\text{trans}j} \) Predicate that must be true of \textbf{App} before passing control back to \textbf{App}

\( \text{Prep}_j \) Action of \textbf{App} causing \( \text{Pre}_{\text{trans}j} \) to be met

\( \text{Trans} \) Action of \textbf{App} effecting functional transition

\( \uparrow\text{Trans} \) Event that \textbf{RM} instructs \textbf{App} to transition to \( S_j \)

\( \downarrow\text{Trans} \) Event that \textbf{App} acknowledges to \textbf{RM} that transition is complete

\( \text{Train}_j \) Action of \textbf{App} initializing \( S_j \) or training its data

\( \uparrow\text{Train}_j \equiv \downarrow\text{Trans} \)

\( \downarrow\text{Train}_j \equiv \uparrow S_j \)

\( \uparrow S_j \) \( S_j \) has now initialized or trained all data and is operational.

---

**Figure 2. Actions and Events in Application and Reconfiguration Mechanism**
Application

- Application consist of modules, each with interface to support reconfiguration assurance
- Properties of the modules compose properties of app
- Each interface function takes a module-specific service level parameter that instructs interface to provide a level of function
- Composition of different module service levels allow system to operate under different specifications

Now we state the necessary design-level properties that must be shown, in order to meet the required high-level reconfiguration properties
Module capabilities

- Interface to the set of functions contained within the module
- Set of possible values for the service level parameter
- Set of persistent data structures
  - Data which is relevant to preconditions, postconditions, and invariants
- Module postcondition
  - Basic coherency condition representing min state requirement for application to continue some form of operation
- Mechanism through which reconfiguration signals are handled
  - Mechanism through which reconfiguration signals are propagated to calling functions
- Set of module transition conditions
- Set of module preconditions
- Mechanism through which a module’s service transition condition is guaranteed to be met
- Timing guarantees on interface functions related to reconfiguration
- Set of assured reconfiguration invariants

Module properties shown using capabilities

| M1 | If none of a module’s functions is currently executing, that module’s postcondition is met. |
| M2 | Each module has a function \texttt{halt} that, when called, will cause its postcondition to be met. |
| M3 | Each module function either: always leaves its state in a consistent state; or when interrupted, calls the module \texttt{halt} function to leave its state consistent with the module postcondition. |
| M4 | If a function is interrupted, its caller is interrupted with no intervening calls to any function other than \texttt{halt}. |
| M5 | There is a method to meet the transition condition for each specification level. |
| M6 | Each function always meets its timing constraint. |
| M7 | The invariant for normal operation is stricter than the generic reconfiguration invariant, which is stricter than the specific reconfiguration invariant: \( I_0 = I_{\text{spec}} = I_{\text{gen}} \). |
Application capabilities

- Modules that compose the system are contained within a separate top-level structure called monitoring layer
- Monitoring layer includes
  - Facility to activate reconfiguration mechanism
  - State variable `config` representing current operation specification
  - Capability to cause operation under current specification
  - Max time `train_time` that training of data might take for each member of `S`

Application properties

<table>
<thead>
<tr>
<th>App1</th>
<th>If <code>App</code> is not reconfiguring, it will function in accordance with the specification represented by <code>config</code>’s value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>App2</td>
<td>Every operation is called from some sequence of functions initiated by the monitoring layer.</td>
</tr>
<tr>
<td>App3</td>
<td>The postcondition is the conjunction of module/postconditions.</td>
</tr>
<tr>
<td>App4</td>
<td>The transition condition is the conjunction of module transition conditions.</td>
</tr>
<tr>
<td>App5</td>
<td>An interrupt of the monitoring layer will cause an immediate transfer of control to reconfig, which is the functional equivalent of the action <code>RM @((\text{Halt}, a) \land @((\text{RM}, b)))</code></td>
</tr>
<tr>
<td>App6</td>
<td><code>config</code>’s value is invariant outside of <code>RM</code>. <code>config = x &lt; @(RM, b), @(\text{RM}, b)</code> ( \Rightarrow ) <code>config = x &lt; @(RM, b), @(\text{RM}, b+1)</code></td>
</tr>
<tr>
<td>App7</td>
<td>There are no circular dependencies among module reinitialization functions.</td>
</tr>
<tr>
<td>App8</td>
<td>If the transition precondition holds at the time the transition is completed, <code>Pre</code> will be met within <code>train_time</code> time units: <code>Pre_{t+\text{train_time}} &lt; @(\text{Train}, b), @(\text{Train}, b) \Rightarrow @(\text{Train}, b) \leq @(\text{Train}, b) + \text{train_time}</code></td>
</tr>
<tr>
<td>App9</td>
<td>The transition takes no real time: <code>@((\text{Trans}, b) = @(\text{Trans}, b) + \text{train_time})</code>. This property is true of the structure rather than a particular application, and can be stated as an axiom.</td>
</tr>
</tbody>
</table>
Reconfiguration Mechanism capabilities

? Implementation of Choose()

? Mechanism through which each module is ordered to meet its precondition for the new service level

Reconfiguration Mechanism properties

RM1 `choose` will be executed immediately when RM is called. `choose` is equivalent to the action Choose. This means that `@((RM, b)) = @((Choose, b))`.

RM2 If the postcondition of choose is met, the new operational specification is the correct one and is stored in `config`.

RM3 If `config`'s value is invariant outside of RM, then `config`'s value is invariant outside of Choose:

\[
\text{App6} \Rightarrow \text{config} \times (x < @((\text{Choose, b}), @((\text{Choose, b}) > \Rightarrow \text{config} \times x < @((\text{Choose, b}), @((\text{Choose, b+1}) > )}
\]

RM4 App7 ⇒ RM calls all the prep functions of the modules in an order in which no dependencies are violated.

RM5 Exactly the prep functions are called between choose and transition to the monitoring layer; this implies:

\[
@((\text{Choose, b}) = @((\text{Prep, c}) \leq @((\text{Prep, c}) + @((\text{Train, b}) \text{ and}
\]

\[
\text{Post} @((\text{Prep, c}), @((\text{Prep, c}) > \Rightarrow \text{RM6} > \text{App6}) \Rightarrow \text{RM7} @((\text{Prep, c}) >
\]

RM6 If each function meets its timing constraint, then App can halt, RM can execute, and App can train within the allotted time:

\[
M6 \Rightarrow @((\text{Halt, a}) + @((\text{LRM, b}) + @((\text{RM, b}) + @((\text{Train, c}) \Rightarrow @((\text{Train, c}) \leq T_i)
\]

RM7 The invariants that must hold during transition hold at the appropriate times:

\[
\text{Inv}_{i,j} < @((\text{Halt, a}), @((\text{Train, a}) \Rightarrow \text{Inv}_{i,j} < @((\text{Train, a}), @((\text{Train, b}) \Rightarrow \text{Inv}_{i,j} < @((\text{Train, b}) @((\text{Train, c}) \Rightarrow
\]

RM8 RM begins before transition and ends at the time of transition; training begins at the time of transition:

\[
@((\text{RM, b}) \leq @((\text{Train, b}) \Rightarrow @((\text{LRM, b}) = @((\text{Train, b})
\]
Now they show, using design-level properties previously stated, that a system using their architecture and satisfying architectural properties will cause high-level properties to hold.

**P1.** $\mathcal{O}(\uparrow R, b) = \mathcal{O}(\downarrow S, a)$

*(R begins at the same time the system is no longer operating under $S_0$)*

P1 is definitional only, and does not impose specific requirements.

**P2.** $\mathcal{O}(\downarrow R, b) = \mathcal{O}(\uparrow S, c)$

*(R ends at the same time the system becomes compliant with $S_j$)*

This property is definitional and so by itself requires no proof. It implies that the system must at some point transition to $S_j$, but in our model this property is subsumed by P6 since $\text{Pre}_i$ can in general be satisfied only after a functional reconfiguration takes place.

**P7.** $\mathcal{O}(\downarrow S, c-1) < \mathcal{O}(\uparrow R, b) \Rightarrow \mathcal{O}(\uparrow S, c) = \mathcal{O}(\downarrow R, b)$

*(The lifetime of R is bounded by any two occurrences of the same specification)*

P7 is definitional only, and does not impose specific requirements.
**P4.** \( \langle \downarrow R, b \rangle - \langle \uparrow R, b \rangle \leq T_{ij} \)

*(R takes less than or equal to \( T_{ij} \) time units)*

This property can be shown using P1, P2, the definitions of Halt and Train, App5, RM8, and RM6.

**P5.** Inv\(_j\) < \( \langle \uparrow R, b \rangle \), \( \langle \downarrow R, b \rangle \) >

*(The transition invariant holds during R)*

This property can be shown using P1, P2, M7, and RM7.
P5

M7 The invariant for normal operation is stricter than the generic reconfiguration invariant, which is stricter than the specific reconfiguration invariant: Inv$_n$ = Inv$_g$ = Inv$_i$

RM7 The invariants that must hold during transition hold at the appropriate times:

\[ \text{Inv}_i < @\text{(Halt, a)} \land @\text{(Halt, a)} > \land \text{Inv}_q < @\text{(Halt, a)} \land @\text{(Choose, b)} > \land \text{Inv}_q < @\text{(Choose, b)} \land @\text{(TS, c)} > \]

P6. Pre$_j$ < @\text{(R, b)} \land @\text{(R, b)} >

(The precondition for $S_j$ is true at the time $R$ ends)

The proof of P6 is more complex because it requires that a series of predicates be satisfied. The proof is aided by using the following lemmas:

L6.1. An interruption will cause the application to meet its postcondition: $\exists t$: time s.t. $a = \text{Post}_i$ < $t$, $t >$

This lemma can be shown through induction using App2, App3, M1, M2, M3, and M4.

L6.2. An interruption will cause control to be transferred back to the monitoring layer: $\exists t$: time s.t. $a = \text{Halt}_j$, $t >$

This lemma can be shown through induction using Ann2 and M4.
Together with a second application of M1, these lemmas imply that an interruption will cause control to be transferred to the monitoring layer at the same time the postcondition is met. This being true,

L6.3. Post < @([RM], b), @(↑Prep_1, c)>

which follows from App5, RM1, RM5, and M1. Using RM5 again, then App9 and App8, we see that at some time t between @([Trans], b) and @(↑Trans, b) + train_time, Prep_1 is satisfied. Because RM2 and RM3 tell us that config holds the correct value, and RM5 and App9 tell us that at ↓Trans control is passed back to the monitoring layer, and using App1 this means that the system is in functional compliance with S_j, we know that App is operating according to S_j, so t = 1S_j; and using P2, P6 is shown.

P3. ∃ t: time s.t. @([R], b) ≤ t ≤ @([R], b) ∧
(Choose(S_i, Env(t)) = S_j)<t, t>

(S_j is the proper choice for the target specification at some point during R)

We present the full proof of P3 to give the reader an example of their construction, and then outline subsequent proofs so that the reader can construct the full chain of reasoning for himself.

For brevity, we write (Choose(S_i, Env(t)) = S_j)<t, t > as the predicate valid(t). We first establish that valid(t) is true for the time Choose ends: valid([@([Choose, b)], RM1 says that

choose ≡ Choose

⇒ (by the relationship we have assigned functional and sequence properties)

choose.post < @([Choose, b], @(↓Choose, b)>

⇒ (RM2)

(Choose(S_i, Env(t)) = S(config)) <@([Choose, b],
@([Choose, b])>
Example: RIPS system

- NASA’s Runway Incursion Prevention System
- Runs on aircraft to prevent collisions with objects on runway - gives advice to pilots
- Specific to RIPS, the Runway Safety Monitor (RSM) algorithm
Service specifications

- Their model of the algo contains 4 major operational specifications
  - S1 - monitors runway and surroundings
  - S2 - monitors runway (easier than S1)
  - S3 - halts and alerts pilots
  - S4 - gives aircraft command to climb and alerts pilot

Main modules

- GEOM - computes basic geometric functions
- IZ - sets up geometry specific to RSM
- ALG - analyzes incoming data

- GEOM is protected by a layer that checks its outputs to give them strong correctness arguments
- Two sets of persistent data
  - Incursion zone structure (belongs to IZ)
  - Data structures for system interface (ALG)
In order to indicate how all of the high-level reconfiguration properties can be proved, they choose 3 representative design-level properties.

The representative module property is M1: If none of a module's functions is currently executing, that module's postcondition is met. Disallow data structure access through any function outside the module interface. Any failure of a check causes reconfiguration signal to propagate through modules. Modules will cause data structures to meet their postconditions.