Assured Reconfiguration of Embedded Real-Time Software

By E. Strunk and J. Knight

Cut and paste job presented by Foo
without permission of authors

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 Si and begins operation under a different target specification Sj*

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
- $Env(t) \rightarrow e \in E$: function that returns the value of the environmental state at time t
- $Choose(S_x, e \in E) \rightarrow S_y$: function that returns appropriate target elements of S, given source elements of S and E.

More terms used

- Pre_x: the precondition that must be satisfied for the system to operate under S_x
- Inv_{ij}: 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 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. $@(\uparrow \boldsymbol{R}, b) = @(\downarrow S_i, a)$

(*$\boldsymbol{R}$ 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. $@(\downarrow \boldsymbol{R}, b) = @(\uparrow S_j, c)$

(*$\boldsymbol{R}$ ends at the same time the system becomes compliant with* S_j)

P3. $\exists$ t: time s.t. $@(\uparrow \boldsymbol{R}, b) \leq t \leq @(\downarrow \boldsymbol{R}, b) \wedge$
$(Choose(S_i, Env(t)) = S_j) < t, t >$

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

Continued

P4. $@(\downarrow \boldsymbol{R}, b) - @(\uparrow \boldsymbol{R}, b) \leq T_{ij}$

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

P5. $Inv_{ij} < @(\uparrow \boldsymbol{R}, b), @(\downarrow \boldsymbol{R}, b) >$

(*The transition invariant holds during* ***R***)

P6. $Pre_j < @(\downarrow \boldsymbol{R}, b), @(\downarrow \boldsymbol{R}, b) >$

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

P7. $@(\downarrow S_j, c\text{-}1) < @(\uparrow \boldsymbol{R}, b) \Rightarrow @(\uparrow S_j, c) = @(\downarrow \boldsymbol{R}, b)$

(*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,…)

App	Action representing operation of the application. Operates continuously throughout ***R***.
RM	Action representing operation of the reconfiguration mechanism.
↑RM	Event that **App** signals fault at top level
↓RM	≡ **↓Trans** (below)
↓S_i	Event that a reconfiguration signal is sent to S_i
Halt	Action of **App** causing S_i to meet **Post**
↑Halt	≡ **↓S_i**
Post	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.
Choose	Action of **RM** determining member of S for S_j

More building blocks

Pre_{transj}	Predicate that must be true of **App** before passing control back to **App**
$Prep_j$	Action of **App** causing **Pre_{transj}** to be met
Trans	Action of **App** effecting functional transition
↑Trans	Event that **RM** instructs **App** to transition to S_j
↓Trans	Event that **App** acknowledges to **RM** that transition is complete
$Train_j$	Action of **App** initializing S_j or training its data
↑$Train_j$	≡ **↓Trans**
↓$Train_j$	≡ **↑S_j**
↑S_j	S_j has now initialized or trained all data and is operational.

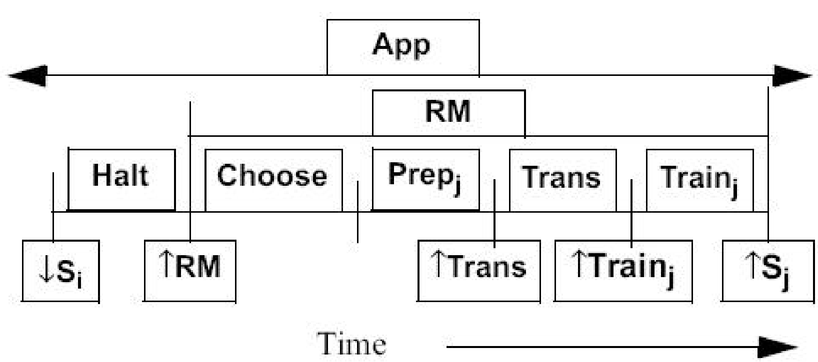

Figure 2. Actions and Events in Application and Reconfiguration Mechanism

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 halt that, when called, will cause its postcondition to be met.
M3	Each module function either: always leaves its data in a consistent state; or when interrupted, calls the module halt function to leave the 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 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: $Inv_i \Rightarrow Inv_{rg} \Rightarrow Inv_{ij}$

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

App1	If **App** is not reconfiguring, it will function in accordance with the specification represented by config's value.
App2	Every operation is called from some sequence of functions initiated by the monitoring layer.
App3	The postcondition is the conjunction of module postconditions.
App4	The transition condition is the conjunction of module transition conditions.
App5	An interrupt of the monitoring layer will cause an immediate transfer of control to reconfig, which is the functional equivalent of the action **RM**: $@(\downarrow Halt, a) = @(\uparrow RM, b)$
App6	config's value is invariant outside of RM: $config = x < @(\downarrow RM, b), @(\downarrow RM, b)> \Rightarrow config = x < @(\downarrow RM, b), @(\uparrow RM, b+1) >$
App7	There are no circular dependencies among module reinitialization functions.
App8	If the transition precondition holds at the time the transition is completed, Pre_j will be met within train_time(j) time units: $Pre_{transj} < @(\uparrow Train_j, b), @(\uparrow Train_j, b) > \Rightarrow @(\downarrow Train_j, b) \leq @(\uparrow Train_j, b) + train\_time(j)$
App9	The transition takes no real time: $@(\uparrow Trans, b) = @(\downarrow Trans, b)$ This property is true of the structure rather than a particular application, and can be stated as an axiom.

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 $@(\uparrow RM, b) = @(\uparrow Choose, b)$.
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: $App6 \Rightarrow (\, config = x < @(\downarrow Choose, b), @(\downarrow Choose, b) > \Rightarrow config = x < @(\downarrow Choose, b), @(\uparrow Choose, b+1) >)$
RM4	$App7 \Rightarrow$ 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: $@(\downarrow Choose, b) = @(\uparrow Prep_j, c) \leq @(\downarrow Prep_j, c) = @(\uparrow Trans, b)$ and $(Post< @(\uparrow Prep_j, c), @(\uparrow Prep_j, c) > \wedge RM4 \wedge App4) \Rightarrow Pre_{transj} < @(\downarrow Prep_j, c), @(\downarrow Prep_j, c) >$
RM6	If each function meets its timing constraint, then App can halt, RM can execute, and App can train within the allotted time: $M6 \Rightarrow @(\downarrow Halt, a) - @(\uparrow Halt, a) + @(\downarrow RM, b) - @(\uparrow RM, b) + @(\downarrow Train_j, c) - @(\uparrow Train_j, c) \leq T_{ij}$
RM7	The invariants that must hold during transition hold at the appropriate times: $Inv_i < @(\uparrow Halt, a), @(\uparrow Halt, a) > \wedge Inv_{ix} < @(\uparrow Halt, a), @(\downarrow Choose, b) > \wedge Inv_{ij} < @(\downarrow Choose, b), @(\uparrow S_j, c) >$
RM8	RM begins before transition and ends at the time of transition; training begins at the time of transition: $@(\uparrow RM, b) \leq @(\uparrow Trans, b) = @(\downarrow RM, b) = @(\uparrow Train_j, b)$

? 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. $@(\uparrow \boldsymbol{R}, b) = @(\downarrow S_i, a)$

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

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

P2. $@(\downarrow \boldsymbol{R}, b) = @(\uparrow S_j, c)$

(***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 Pre_j can in general be satisfied only after a functional reconfiguration takes place.

P7. $@(\downarrow S_j, c-1) < @(\uparrow \boldsymbol{R}, b) \Rightarrow @(\uparrow S_j, c) = @(\downarrow \boldsymbol{R}, b)$

(*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. $@(\downarrow \boldsymbol{R}, b) - @(\uparrow \boldsymbol{R}, b) \leq T_{ij}$

(***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. $Inv_{ij} < @(\uparrow \boldsymbol{R}, b), @(\downarrow \boldsymbol{R}, b) >$

(*The transition invariant holds during* ***R***)

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

P4

P1. $@(\uparrow R, b) = @(\downarrow S_i, a)$

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

P2. $@(\downarrow R, b) = @(\uparrow S_j, c)$

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

App5	An interrupt of the monitoring layer will cause an immediate transfer of control to reconfig, which is the functional equivalent of the action **RM**: $@(\downarrow Halt, a) = @(\uparrow RM, b)$
RM6	If each function meets its timing constraint, then App can halt, RM can execute, and App can train within the allotted time: $M6 \Rightarrow @(\downarrow Halt, a) - @(\uparrow Halt, a) + @(\downarrow RM, b) - @(\uparrow RM, b) + @(\downarrow Train_j, c) - @(\uparrow Train_j, c) \leq T_{ij}$
RM8	RM begins before transition and ends at the time of transition; training begins at the time of transition: $@(\uparrow RM, b) \leq @(\uparrow Trans, b) = @(\downarrow RM, b) = @(\uparrow Train_j, b)$

$\downarrow$**S**$_i$	Event that a reconfiguration signal is sent to S_i
Halt	Action of **App** causing S_i to meet **Post**
$\uparrow$**Halt**	$\equiv \downarrow$**S**$_i$

Trans	Action of **App** effecting functional transition
$\uparrow$**Trans**	Event that **RM** instructs **App** to transition to S_j
$\downarrow$**Trans**	Event that **App** acknowledges to **RM** that transition is complete
Train$_j$	Action of **App** initializing S_j or training its data
$\uparrow$**Train**$_j$	$\equiv \downarrow$**Trans**
$\downarrow$**Train**$_j$	$\equiv \uparrow$**S**$_j$

P5

M7 The invariant for normal operation is stricter than the generic reconfiguration invariant, which is stricter than the specific reconfiguration invariant: $Inv_i \Rightarrow Inv_{ix} \Rightarrow Inv_{ij}$

RM7 The invariants that must hold during transition hold at the appropriate times:
Inv_i < @(↑Halt, a), @(↑Halt, a) > ∧ Inv_{ix} < @(↑Halt, a), @(↓Choose, b) > ∧ Inv_{ij} < @(↓Choose, b), @(↑S_j, c) >

P6. Pre_j < @(↓***R***, b), @(↓***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: ∃ t: time s.t. @(↓S_i, a) = t ⇒ ∃ t: time s.t. Post < 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: ∃ t: time s.t. @(↓S_i, a) = t ⇒ ∃ t: time s.t. t = @(↓Halt, a).
This lemma can be shown through induction using App2 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 $< @(\uparrow \mathrm{RM}, b), @(\uparrow \mathrm{Prep}_j, c) >$

which follows from App5, RM1, RM5, and M1. Using RM5 again, then App9 and App8, we see that at some time t between $@(\downarrow \mathrm{Trans}, b)$ and $@(\downarrow \mathrm{Trans}, b)$ + train_time, Pre_j 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 $t = \uparrow S_j$; and using P2, P6 is shown.

P3. $\exists$ t: time s.t. $@(\uparrow \boldsymbol{R}, b) \le t \le @(\downarrow \boldsymbol{R}, b) \wedge$
$(\mathrm{Choose}(S_i, \mathrm{Env}(t)) = S_j)< t, t >$

(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 $(\mathrm{Choose}(S_i, \mathrm{Env}(t)) = S_j)< t, t >$ as the predicate validj(t). We first establish that validj(t) is true for the time Choose ends: validj(@(↓Choose, b)). RM1 says that

choose $\equiv$ Choose

$\Rightarrow$ (by the relationship we have assigned functional and sequence properties)

choose.post $< @(\downarrow \mathrm{Choose}, b), @(\downarrow \mathrm{Choose}, b) >$

$\Rightarrow$ (RM2)

$(\mathrm{Choose}(S_i, \mathrm{Env}(t)) = S(\mathrm{config})) <@(\downarrow \mathrm{Choose}, b), @(\downarrow \mathrm{Choose}, b)>$

App6 and RM3 tell us that (config = j) < @(↓Choose, b), @(↓Choose, b) > since S_{config} will be the new operational specification and config will remain constant until $\uparrow S_j$. Together with the above statement we see

validj(@(↓Choose, b)).

Intuitively, it is obvious that @(↑*R*, b) ≤ @(↓Choose, b) ≤ @(↓*R*, b). More rigorously:

∃ t: time s.t. @(↓Choose, b) = t ∧ validj(t)

⇒ (axiomatic basis of time in RTL)

∃ t: time s.t. @(↑Choose, b) ≤ t ≤ @(↓Choose, b) ∧ validj(t)

⇔ (RM1 and RM5)

∃ t: time s.t. @(↑RM, b) ≤ t ≤ @(↑Trans, b) ∧ validj(t)

⇔ (App5 and RM8)

∃ t: time s.t. @(↓Halt, a) ≤ t ≤ @(↑$Train_j$, c) ∧ validj(t)

⇒ (axiomatic basis of time in RTL)

∃ t: time s.t. @(↑Halt, a) ≤ t ≤ @(↓$Train_j$, c) ∧ validj(t)

⇒ (definitions of Halt and Train)

∃ t: time s.t. @(↓S_i, a) ≤ t ≤ @(↑S_j, c) ∧ validj(t)

⇒ (P1 and P2, substituting for validj(t))

∃ t: time s.t. @(↑*R*, b) ≤ t ≤ @(↓*R*, b) ∧ (Choose(S_i, Env(t)) = S_j)< t, t >

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