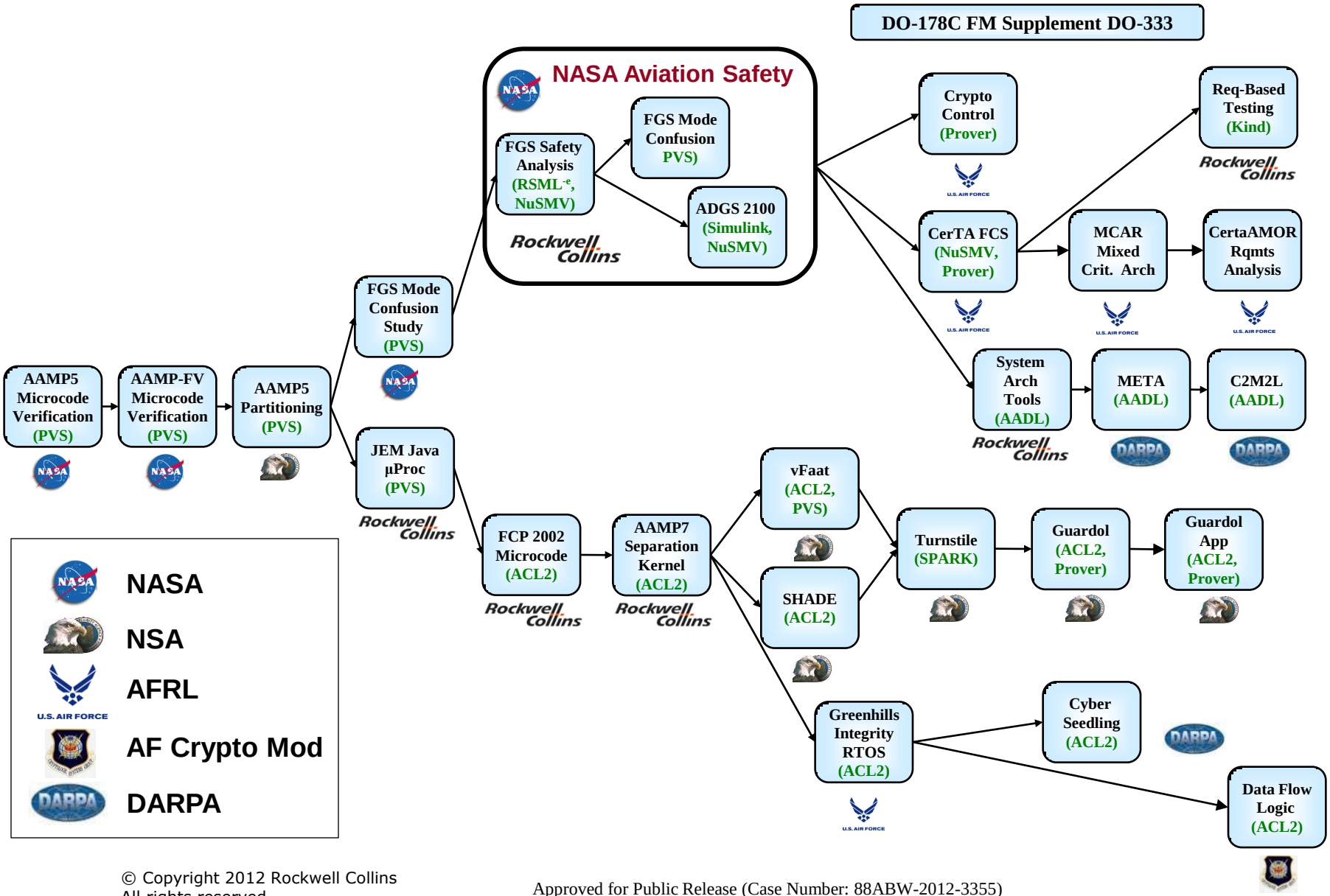
CertaAMOR: Automated Modeling of Requirements

Lucas Wagner
Rockwell Collins
lgwagner@rockwellcollins.com
319-295-5672

Rockwell Collins

Approved for Public Release (Case Number: 88ABW-2012-3355)

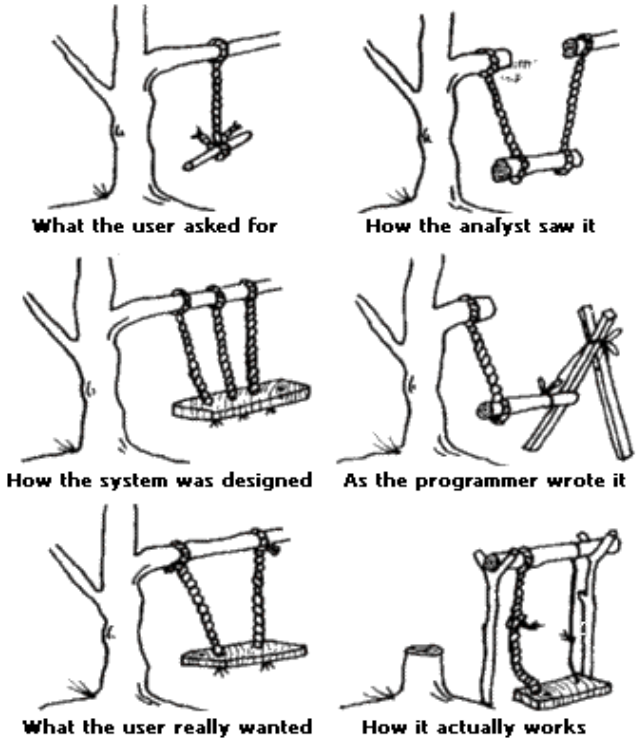
Formal Methods At Rockwell Collins



Approved for Public Release (Case Number: 88ABW-2012-3355)

Why Analyze Requirements?

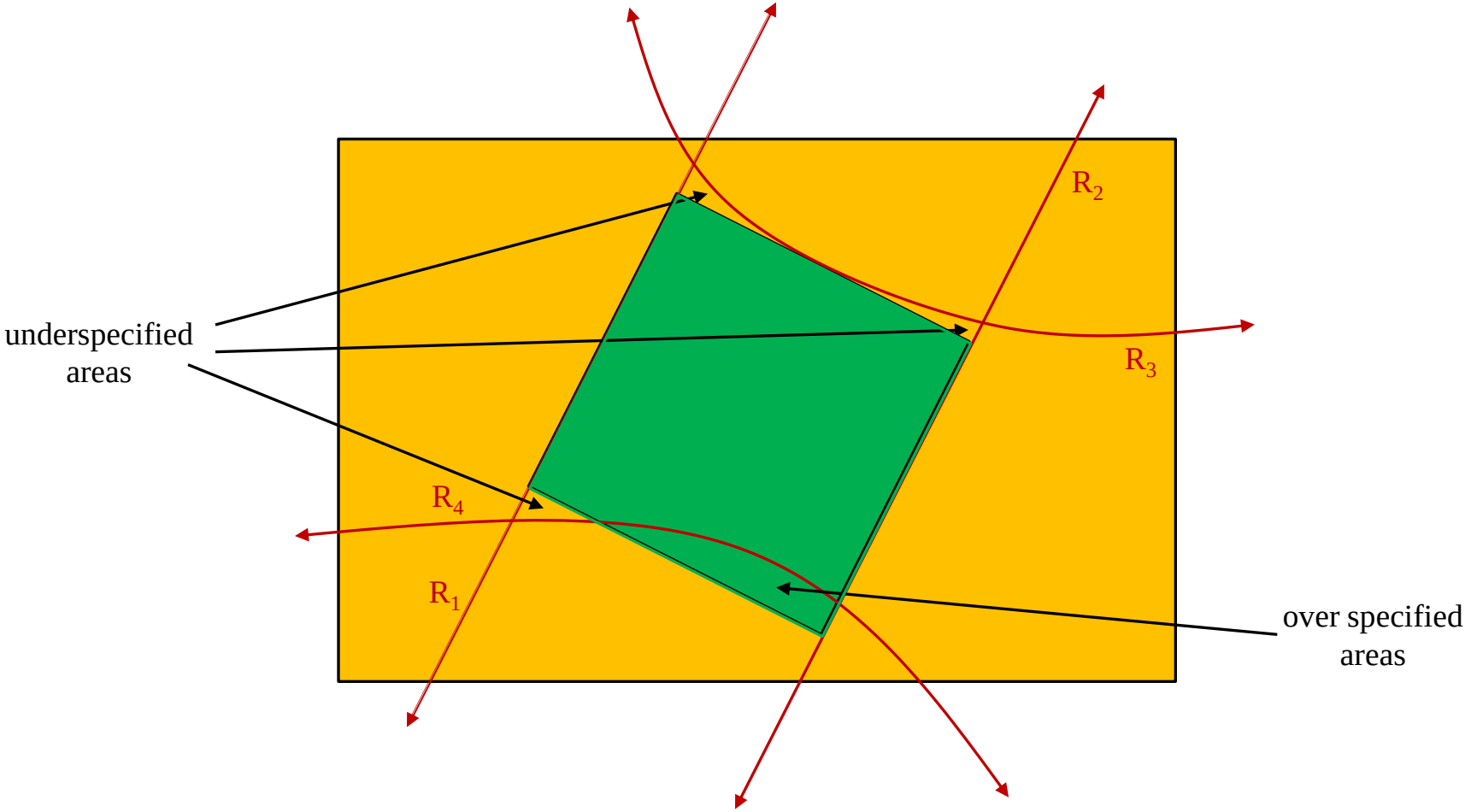
Requirements are the main vehicle through which we describe the desired behaviors of complex, critical systems that we develop.



Approved for Public Release (Case Number: 88ABW-2012-3355)

Flaws in the Traditional Specification Process

The way we currently specify system behaviors lacks focus on properly validating our system's behaviors. The graphic below illustrates the problem:



Approved for Public Release (Case Number: 88ABW-2012-3355)

The Approach

For this effort we are making use of existing methods and tools to perform early requirements analysis of complex systems. The approach is two-pronged:

- Requirements analysis is performed on a set of formalized requirements using the Requirements Analysis Tool (RAT) developed at Fondazione Bruno Kessler in the EU.

- English language requirements are formalized using a domain specific language that implements the Specification Patterns approach introduced by Matt Dwyer et al. at Kansas State University in the late ‘90s/early 2000s.

Approved for Public Release (Case Number: 88ABW-2012-3355)

First Attempt

As a first step, we analyzed the requirements of our microwave oven training example. We chose this example because it is reasonably complex and had existing formalized requirements.

Analyzing the microwave requirements in the RAT tool involved the following steps:

- Describing the interface of the microwave
- Identifying and formalizing the "validation properties" or the behaviors we want our system to exhibit.
- Specifying the requirements of the microwave using Linear Temporal Logic (LTL).
- Iteratively analyzing the requirements (and making fixes) until correct.

Approved for Public Release (Case Number: 88ABW-2012-3355)

Microwave mode logic interface

Takes input from the keypad.

- start, clear, steps_to_cook

Reads sensors.

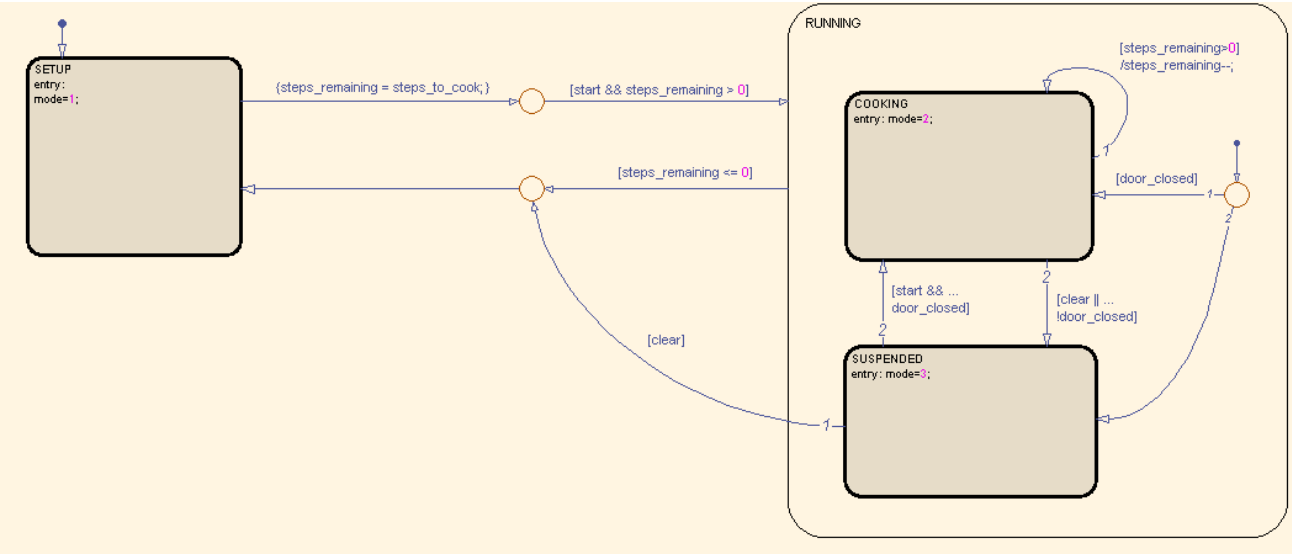
- door_closed

Computes output data for the user.

- steps_remaining

Produces an output that controls the microwave

- Mode (SETUP, COOKING, SUSPENDED)



Approved for Public Release (Case Number: 88ABW-2012-3355)

Validation Properties

We have previously identified several properties of the microwave that can be used to validate that our requirements provide the correct behavior. They are:

- G(COOKING -> DOOR_CLOSED)
- G(RUNNING -> STEPS_REMAINING > 0)
- G(RUNNING -> X(RUNNING -> (STEPS_REMAINING <= PREV_STEPS_REMAINING)))
- G(COOKING -> F(!COOKING))

Our system requirements should sufficiently constrain the behaviors of the system to enforce these validation properties. In this sense, these properties describe the "green box" from the image before.

Mode Logic Requirements

R1. SETUP & STEPS_REMAINING=0

- *The microwave shall start in SETUP mode.*

R2. G(SETUP -> X((KP_START & STEPS_TO_COOK > 0) -> RUNNING))

- *While in SETUP mode, the microwave shall enter the RUNNING mode if the start key is pressed when the user has entered a nonzero cooking time.*

R3. G(!RUNNING -> X(RUNNING -> (DOOR_CLOSED -> COOKING)))

- *When the microwave enters the RUNNING mode it shall enter the COOKING submode if the door is closed.*

R4. G(!RUNNING -> X(RUNNING -> (!DOOR_CLOSED -> SUSPENDED)))

- *When the microwave enters the RUNNING mode it shall enter the SUSPENDED submode if the door is open.*

R5. G((RUNNING & STEPS_REMAINING <= 0) -> X(SETUP))

- *While in the RUNNING mode the microwave shall enter the SETUP mode when the steps remaining to cook becomes zero or less.*

R6. G(COOKING -> X((!DOOR_CLOSED | KP_CLEAR) -> SUSPENDED))

- *While in the COOKING mode, the microwave shall entered the SUSPENDED mode if the door is opened or the clear key is pressed.*

R7. G(SUSPENDED -> X(KP_CLEAR -> SETUP))

- *While in the SUSPENDED mode, the microwave shall enter the SETUP mode if the clear key is pressed.*

R8. G(SUSPENDED -> X((KP_START & DOOR_CLOSED) -> COOKING))

- *When the microwave enters the RUNNING mode it shall enter the COOKING submode if the door is closed.*

R9. G(SETUP -> STEPS_REMAINING = STEPS_TO_COOK)

- *While in SETUP mode, the steps remaining to cook shall be set to the time entered on the microwave keypad by the user.*

R10. G(RUNNING -> X(RUNNING -> (STEPS_REMAINING <= PREV_STEPS_REMAINING)))

- *While in the RUNNING mode, the steps remaining shall be monotonically decreasing. (The microwave makes progress towards termination)*

R11. G(COOKING -> X(COOKING -> (STEPS_REMAINING = PREV_STEPS_REMAINING−1)))

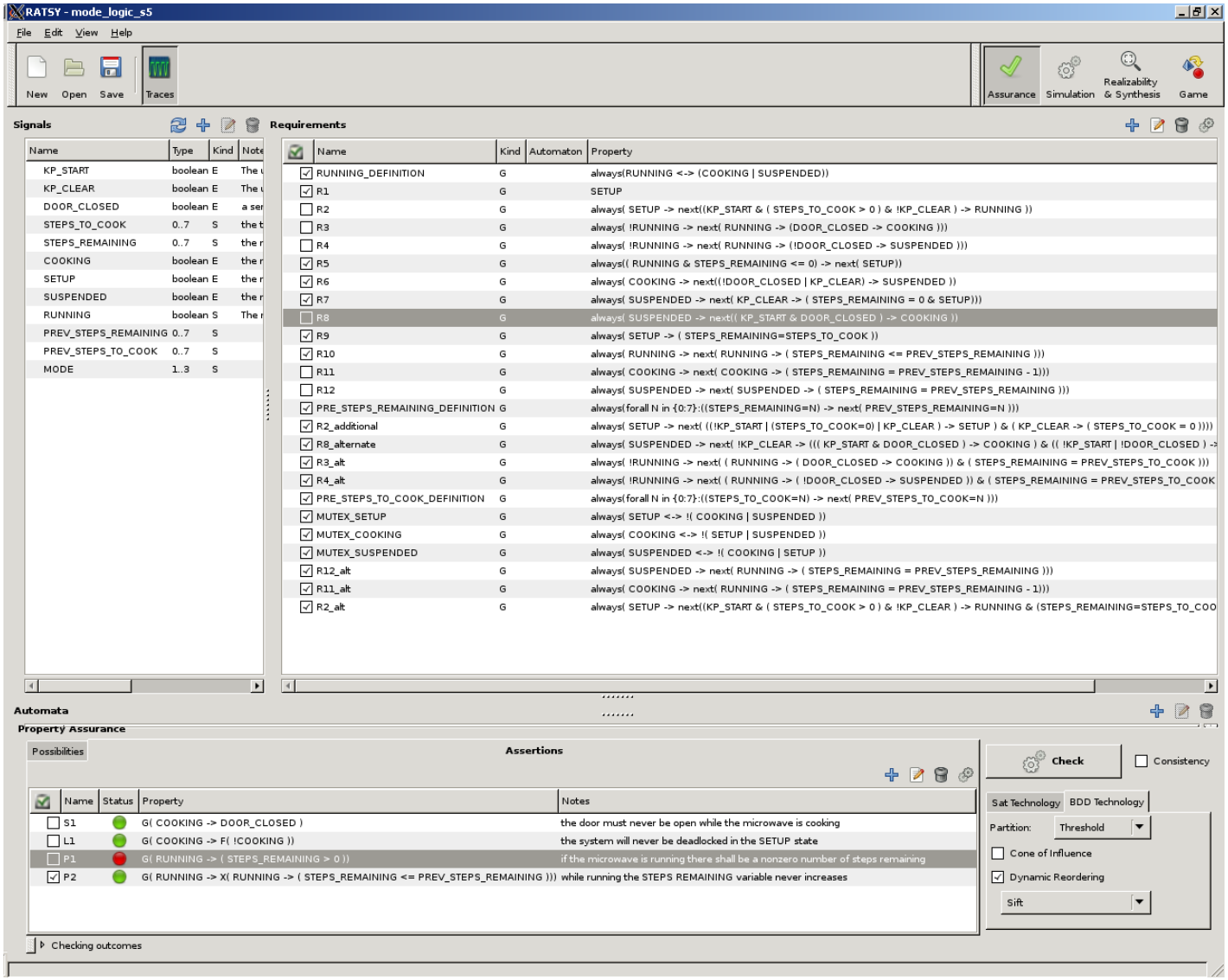
- *While in the COOKING mode the steps remaining shall decrement.*

R12. G(SUSPENDED -> X(SUSPENDED -> (STEPS_REMAINING = PREV_STEPS_REMAINING)))

- *While in the SUSPENDED mode the steps remaining shall not change.*

Rockwell Collins

RAT Tool GUI



Approved for Public Release (Case Number: 88ABW-2012-3355)

What did we find?

We checked our set of 12 requirements against our 4 validation properties.

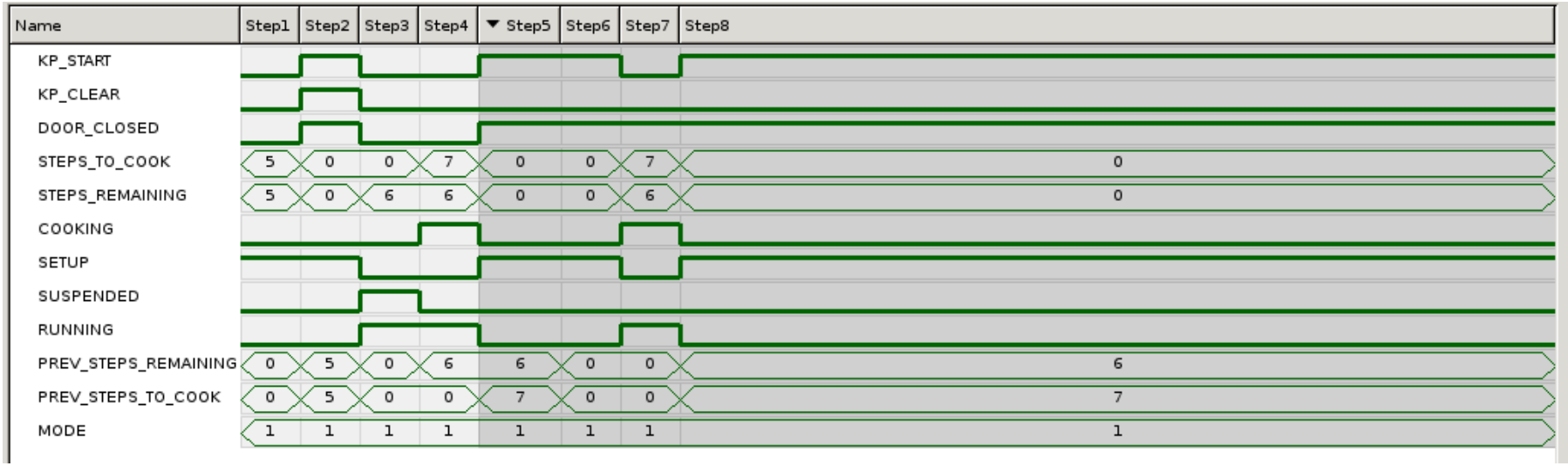
The process involved checking the set of requirements against a single validation property at a time. Each error was iteratively fixed until proven true. At the end, the analysis was run one last time on each property to identify potential regression faults.

In total, we identified 7 changes that had to be made to the requirements to satisfy our validation properties.

The next two slides will detail a two of the errors encountered and suitable fixes.

Approved for Public Release (Case Number: 88ABW-2012-3355)

Requirements Flaw #1



Considering the validation property: G(COOKING -> DOOR_CLOSED)

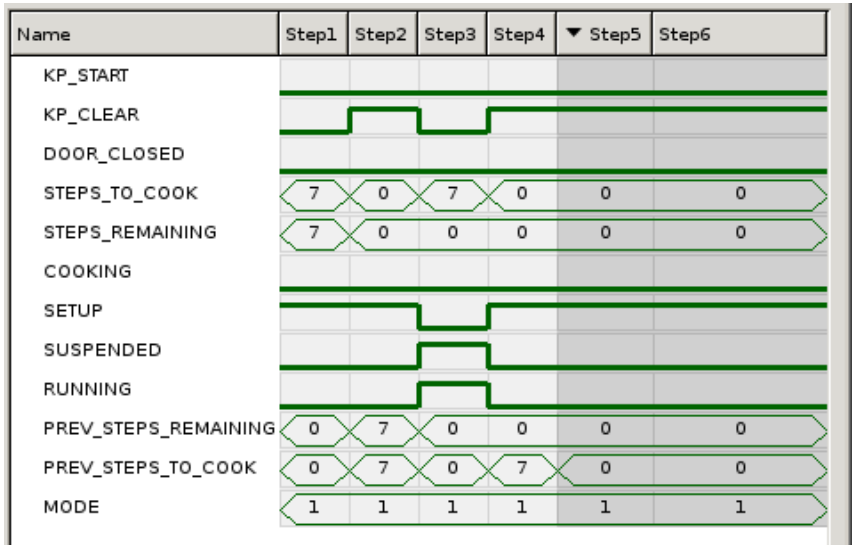
Step 1: Normal initialization
Step 2: Clear button is pressed, zeroing out the steps_to_cook.
Step 3: Transition from Setup to Suspended
Step 4: Transition from Suspended to Cooking with neither the door closed or the start button pressed.

R8. G(SUSPENDED -> X((KP_START & DOOR_CLOSED) -> COOKING))

Requirement R8 is incomplete, allowing mode transitions without user involvement.

Approved for Public Release (Case Number: 88ABW-2012-3355)

Requirements Flaw #2



Considering the validation property: G(RUNNING -> (STEPS_REMAINING > 0))

Step 1: Normal initialization
Step 2: Clear button is pressed, zeroing out steps_to_cook.
Step 3: Transition from Setup to Suspended, however there is no time on the clock.

Requirements R3 and R4 control the way the system enters the Running state, but they do not restrict the values of the "steps_remaining" variable.

R3. G(!RUNNING -> X(RUNNING -> (DOOR_CLOSED -> COOKING)))
R4. G(!RUNNING -> X(RUNNING -> (!DOOR_CLOSED -> SUSPENDED)))
R9. G(SETUP -> STEPS_REMAINING = STEPS_TO_COOK)

What about Production Requirements?

To apply the method on a production system we must address the problem of formalizing English language requirements so we can perform formal analysis.

Rockwell Collins has developed a Domain Specific Language (DSL) for requirements specification that provides tool support and syntactical checking for the work done by Kansas State (Matt Dwyer) in the area specification patterns.

Approved for Public Release (Case Number: 88ABW-2012-3355)

Specification Patterns

Dwyer et al. developed a pattern-based approach to specifying temporal properties.

Each specification pattern contains a scope and predicate pair. This uniquely defines a logical formula and when during program execution it should hold.

Possible scopes that are:

global, **before** **q**, **after** **q**, **between** **[q,r]**, **after** **q** **until** **r**

The types of formulas that are defined are:

always, **existence**, **precedence**, **response**

Together the scope and formula patterns can be used to express a variety of useful temporal relationships over systems.

after (altitude > 1000.0) :: **always** (mode != TAKEOFF)

Approved for Public Release (Case Number: 88ABW-2012-3355)

Microwave Example in SpecDSL

Java - testDSL/microwave.specdsl - Eclipse SDK

File Edit Navigate Search Project Specification Menu Run Window Help

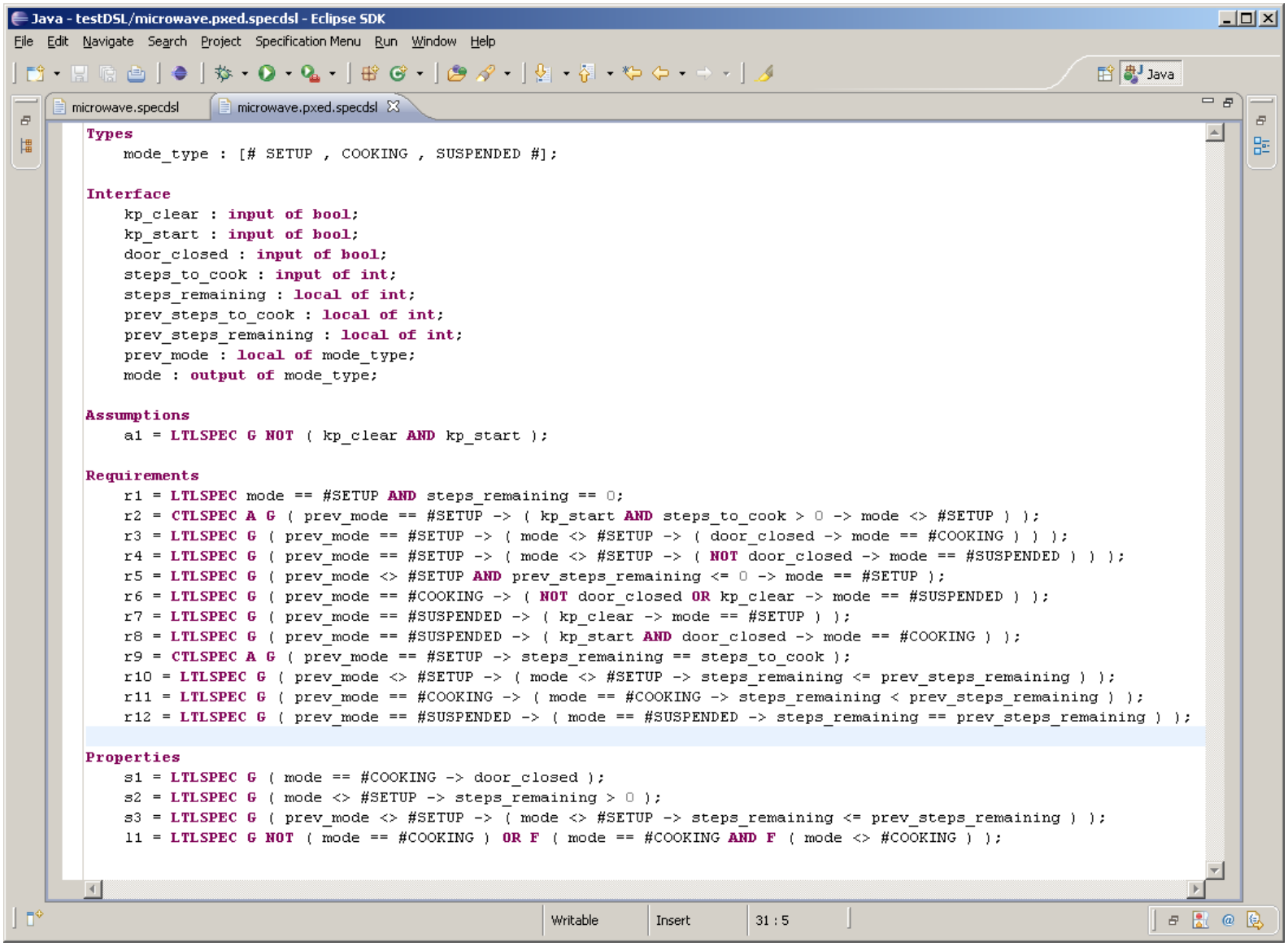
microwave.specdsl | *microwave.pxed.specdsl

```
Types
    mode_type : [# SETUP, COOKING, SUSPENDED #] ;

Interface
    kp_clear      : input of bool ;
    kp_start      : input of bool ;
    door_closed   : input of bool ;
    steps_to_cook : input of int ;
    steps_remaining : local of int ;
    prev_steps_to_cook : local of int ;
    prev_steps_remaining : local of int ;
    prev_mode : local of mode_type ;
    mode : output of mode_type ;

Assumptions
    a1 = global :: always not ( kp_clear and kp_start ) ; --rewrite as LTL

Requirements
    r1 = initial :: ( mode == #SETUP ) and ( steps_remaining == 0 ); -- rewrite as LTL
    r2 = global :: always ( prev_mode == #SETUP ) implies (( kp_start and ( steps_to_cook > 0 )) implies ( mode <> #SETUP )) ; --rewrite as CTL
    r3 = global :: always ( prev_mode == #SETUP ) implies (( mode <> #SETUP ) implies ( door_closed implies ( mode == #COOKING ))) ; --rewrite as LTL
    r4 = global :: always ( prev_mode == #SETUP ) implies (( mode <> #SETUP ) implies ( not door_closed implies ( mode == #SUSPENDED ))) ; --rewrite as LTL
    r5 = global :: always (( prev_mode <> #SETUP ) and ( prev_steps_remaining <= 0 )) implies ( mode == #SETUP ) ; --rewrite as LTL
    r6 = global :: always ( prev_mode == #COOKING ) implies (( not door_closed or kp_clear ) implies ( mode == #SUSPENDED )) ; --rewrite as LTL
    r7 = global :: always ( prev_mode == #SUSPENDED ) implies ( kp_clear implies ( mode == #SETUP )) ; --rewrite as LTL
    r8 = global :: always ( prev_mode == #SUSPENDED ) implies (( kp_start and door_closed ) implies ( mode == #COOKING )) ; --rewrite as LTL
    r9 = global :: always ( prev_mode == #SETUP ) implies ( steps_remaining == steps_to_cook ) ; --rewrite as CTL
    r10 = global :: always ( prev_mode <> #SETUP ) implies (( mode <> #SETUP ) implies ( steps_remaining <= prev_steps_remaining )) ; --rewrite as LTL
    r11 = global :: always ( prev_mode == #COOKING ) implies (( mode == #COOKING ) implies ( steps_remaining < prev_steps_remaining )) ; --rewrite as LTL
    r12 = global :: always ( prev_mode == #SUSPENDED ) implies (( mode == #SUSPENDED ) implies ( steps_remaining == prev_steps_remaining )) ; --rewrite as LTL

Properties
    s1 = global :: always ( mode == #COOKING ) implies door_closed     ; --rewrite as LTL
    s2 = global :: always ( mode <> #SETUP ) implies ( steps_remaining > 0 ) ; --rewrite as LTL
    s3 = global :: always ( prev_mode <> #SETUP ) implies (( mode <> #SETUP ) implies ( steps_remaining <= prev_steps_remaining )) ; --rewrite as LTL
    l1 = after ( mode == #COOKING   ) :: exists ( mode <> #COOKING   ) ; --rewrite as LTL
```

Writable | Insert | 37 : 88

Approved for Public Release (Case Number: 88ABW-2012-3355)

Translated Specification



Approved for Public Release (Case Number: 88ABW-2012-3355)

Future Work

Our next steps will focus on applying the method to an example of industrial size and significance. This will help us better understand issues with the process. Some issues include:

1. What additional constructs will be necessary in the Specification Language to work on an industrial system?

2. What analysis can we do in the Eclipse environment to augment the RAT tool analysis?

3. For large systems how can we reason about the system in pieces and then take that knowledge to say something meaningful about the whole? (Compositional reasoning)

Approved for Public Release (Case Number: 88ABW-2012-3355)

Acknowledgements

Thank you to AFRL for funding this work.

More information on the RAT tool can be found at http://rat.fbk.eu

More information on the Specification Patterns work by Matt Dwyer at Kansas State can be found at: http://patterns.projects.cis.ksu.edu/

More information on the Specification DSL language and related tools can be obtained by contacting myself, Lucas Wagner at lgwagner@rockwellcollins.com.

Approved for Public Release (Case Number: 88ABW-2012-3355)

Questions?

Approved for Public Release (Case Number: 88ABW-2012-3355)