Engineering Large-Scale Software-Intensive Systems

Geoff Dromey
Software Quality Institute
ARC Centre for Complex Systems

Griffith UNIVERSITY

Plato's Advice

"The beginning
is the most important part
of the work"

Applies very much to Systems & Software Engineering

Large Projects … Greatest Risks

- **Failure to squarely address the problems of <u>scale</u> and <u>complexity</u>**
- **Failure to resolve the <u>imperfect knowledge</u> associated with large sets of requirements for systems.**
- **Failure to build the "<u>right</u>" system.**
- **Failure to keep team <u>productive.</u>**

My Last Seven Years …

- Failure to squarely address the problems of <u>scale</u> and <u>complexity</u>

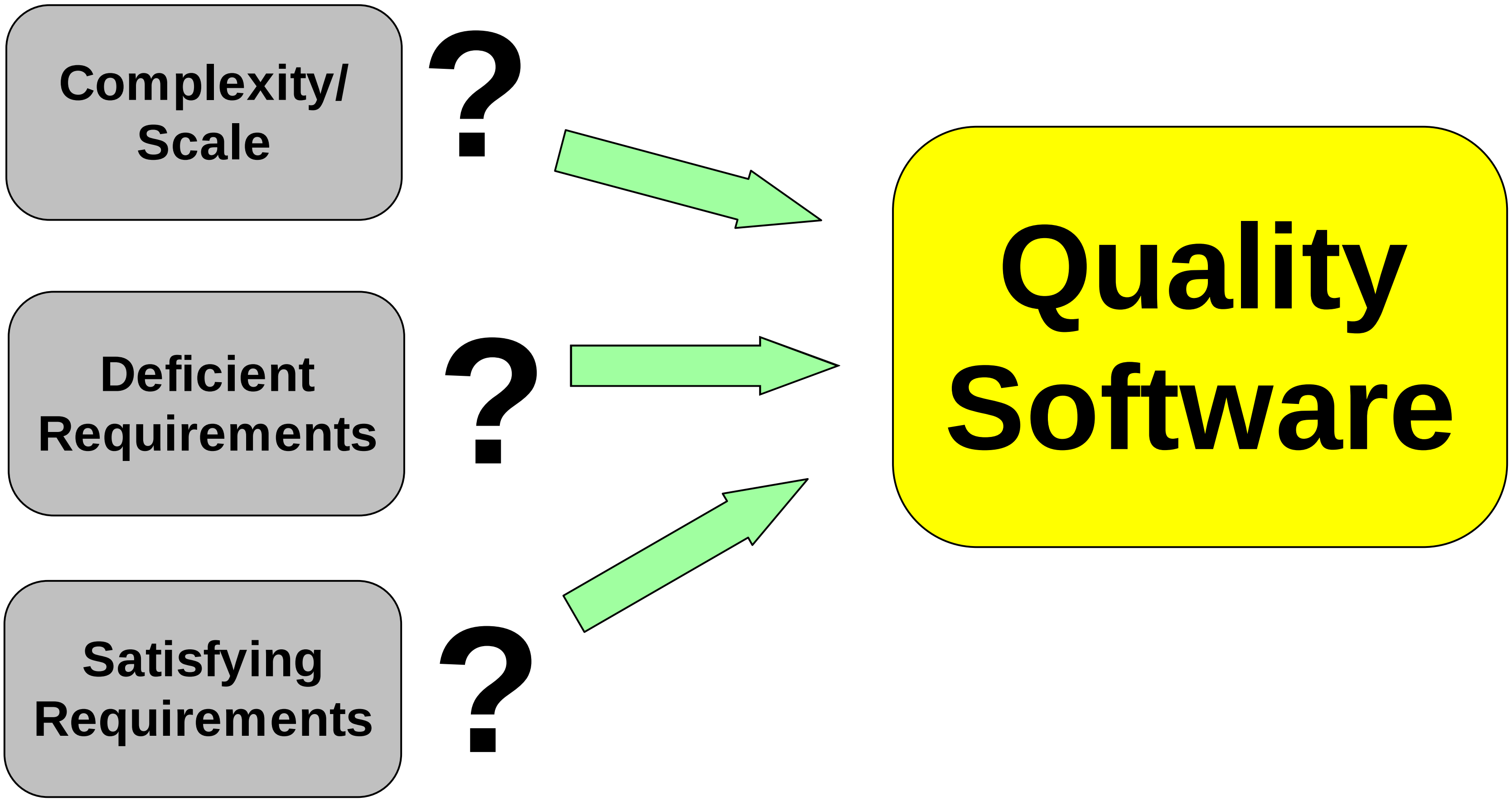
- Failure to resolve the <u>imperfect knowledge</u> associated with large sets of requirements for systems.

- Failure to build the “<u>right</u>” system.

- Failure to keep team <u>productive.</u>

A Search for Repeatabiliy of Outcomes

Threats to Project Success



These Problems are all interdependent

Problem 1 - Complexity

4.4.8.6 Report XXXXXXX

XXXX health information is requested to aid in planning required XXXX maintenance.

CG2) The XCS shall process each HR (XXXX health request) command message received from the YCS.

CG2.1) An HR command message shall be the first message received after the initiation of each "Manage XXXX" transaction.

CG2.2) The XCS may receive a XXXX health request message anytime during a "XXXX " transaction. (Describes order during transaction.)

CG2.3) A XXXX health request command message will only be accepted by the XCS during an active "XXXX " transaction. (Describes condition under which an HR may be received.)

CG3) The XCS shall prepare and send an HA (XXXX health acknowledgment) message to the YCS in response to an HR (XXXX health request) command message.

Thousands of requirements like this

Scale + Interaction => Complexity

Problem 2 – Deficient Requirements

“Deficient Requirements are the single biggest cause of project failure”

feature

requirements

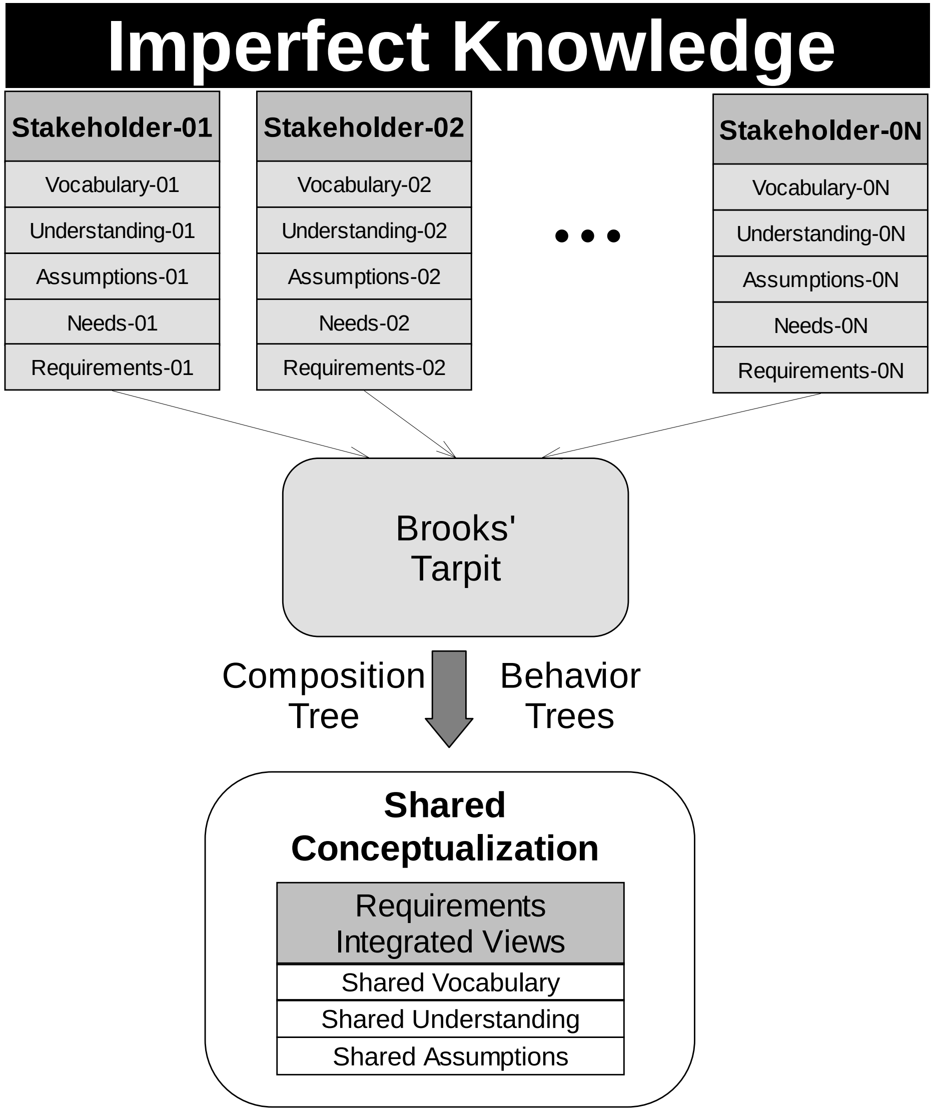
Requirements Engineering as a Success Factor in Software Projects

Hubert F. Hofmann, *General Motors*

Franz Lehner, *University of Regensburg*

Deficient requirements are the single biggest cause of software project failure. From studying several hundred organizations, Capers Jones discovered that RE is deficient in more than 75 percent of all enterprises.[1] In other words, getting requirements right might be the single most important and difficult part of a software project. Despite its importance, we know surprisingly little about the actual process of specifying software. “The RE Process” sidebar provides a basic description.

Problem 2 – Deficient Requirements



Inconsistencies among stakeholders – major issue

Problem 3 - Satisfying Requirements - Example

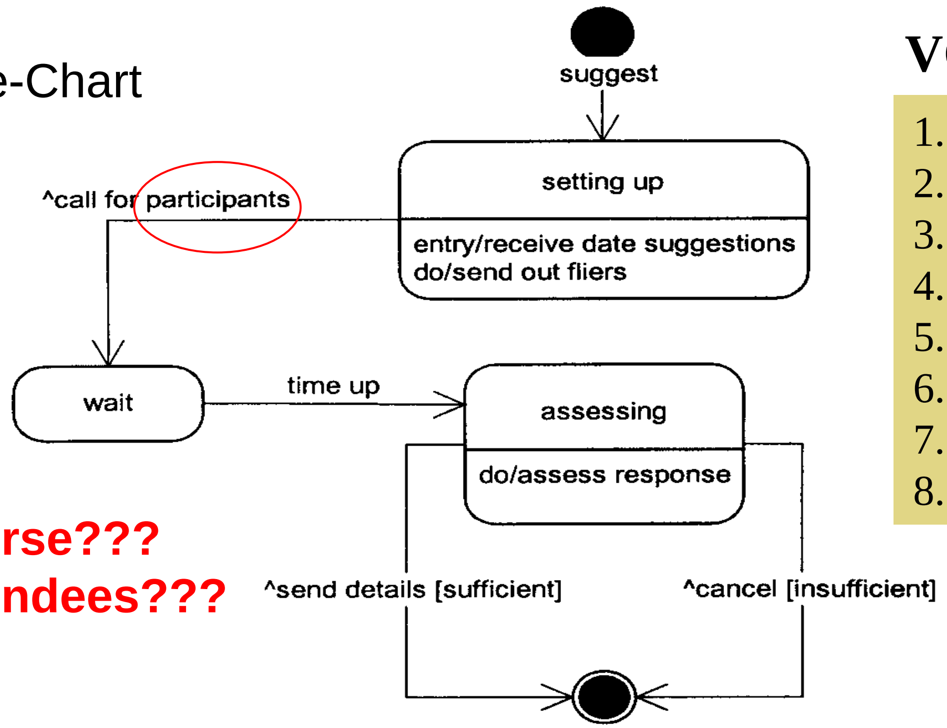
USE-CASE

Receive suggested dates for course
Send out fliers for the course
Wait for a predetermined length of time
When time is up, assess the response
If the response is sufficient, send details to attendees
If response is insufficient, cancel the course

Participants?

Claimed to be “equivalent” to text version on page .226

State-Chart



Course???
Attendees???

VOCABULARY CHANGES

1. “Course” omitted
2. “Participants” introduced
3. “Attendees” omitted
4. “Predetermined length of time” omitted
5. “Setting up” introduced
6. “Entry” introduced
7. “Do” introduced
8. “Call for” introduced

Loss of original intention – major issue

Problem 3 – The "Right" System ?

"The hardest single part of building a software system is deciding what to build, … No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later"

F.P. Brooks

The Greatest Challenge

Scale & Complexity

Imperfect Knowledge

4.4.8.6 Report XXXXXXX

XXXX health information is requested to aid in planning required XXXX maintenance.

CG2) The XCS shall process each HR (XXXX health request) command message received from the YCS.

CG2.1) An HR command message shall be the first message received after the initiation of each "Manage XXXX" transaction.

CG2.2) The XCS may receive a XXXX health request message anytime during a "XXXX " transaction. (Describes order during transaction.)

CG2.3) A XXXX health request command message will only be accepted by the XCS during an active "XXXX " transaction. (Describes condition under which an HR may be received.)

CG3) The XCS shall prepare and send an HA (XXXX health acknowledgment) message to the YCS in response to an HR (XXXX health request) command message.

Thousands of requirements like this

Large Teams Involved

?

The Right System

Current Software Engineering - Strategy

Build system to <u>satisfy</u> the requirements

Example: “Catalysis” – D’Souza & Wills (1999)

- Imply: other methods require a “miracle” to go from requirements to code (p.611)
- Claim: “Catalysis reduces such magic” but you need to read 688 pages to find out
- Advocate: “Treat your system as a single object, define the type of any system implementation that would meet the requirements” (p.596)

Comment – Perhaps a “smaller” miracle

Current Software Engineering - Strategy

Build system to <u>satisfy</u> the requirements

This is TOO Hard

Current SE Methods – Fail to consistently deliver

Recent Failures – June Verner (NICTA)

Most organizations try to hide their failures

A recent "Hall of Shame" IEEE Spectrum, Nov. 2005:

	(in $US millions)
FBI	100
UK Inland Revenue	33
Ford Motor Company	400
Sainsbury's	527
Sydney Water Corp	32.2

Recent Australian problems include:

- National Australia Bank's ERP project,
- RMIT's Academic Management System,
- Victoria State's Infrastructure Management System,
- Federal Government's new sea cargo import reporting system
- But there are many more.......

Current methods buckle under complexity

We need an Alternative way of Thinking

A Lesson From Mathematics

"All mathematics exhibits in its conclusions only what is already implicit in its premises -- all mathematical derivation can be viewed simply ***as change in representation, making evident what was previously true but obscure****."*

- Herbert Simon

Treat Requirements like Premises

Alternative Way of Thinking

Build system
OUT OF
its requirements

REPRESENTATION – is the key to doing this

Alternative Way of Thinking

Build system
OUT OF
its requirements

Implies can deal with
ONE Requirement
At a time

REPRESENTATION – is the key to doing this

Behavior Engineering

Tackling Complexity Head-on

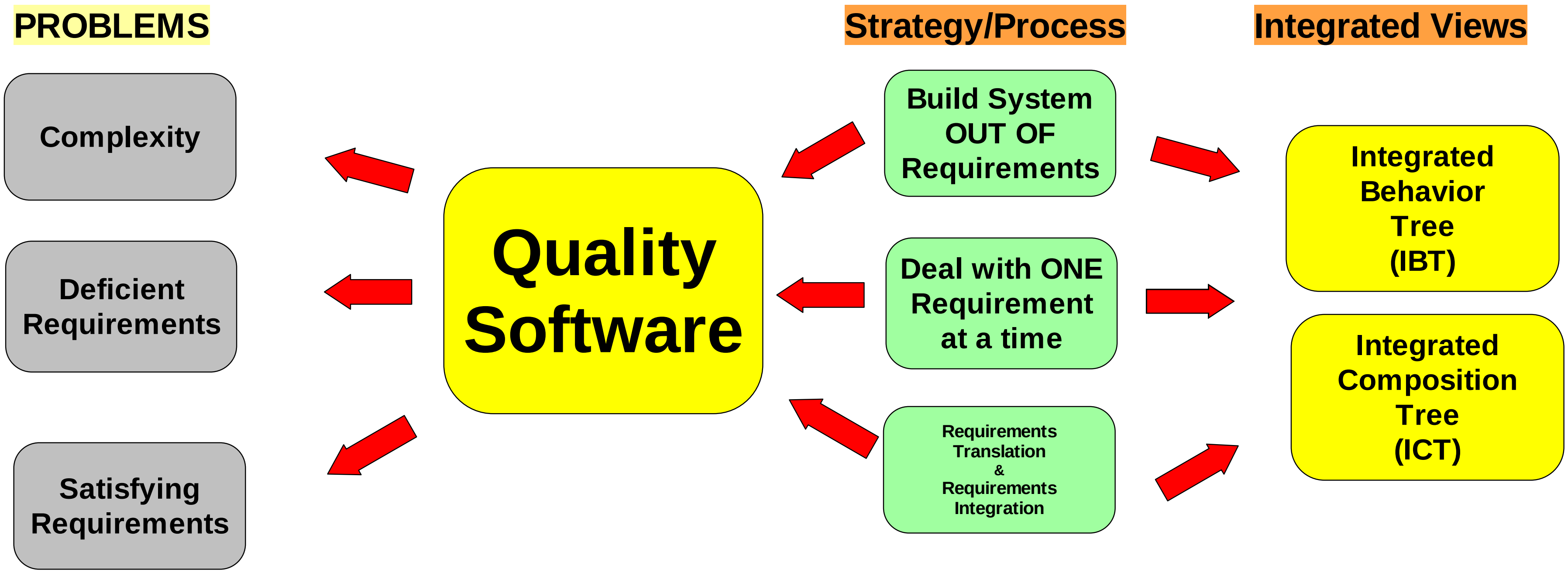
Behavior Engineering

Tackling Verification & Validation Head-on

Behavior Engineering

Tackling Deficient Requirements Head-on

Behavior Engineering



Tackling Software Engineering’s Problems Head-on

Tackling Complexity

The stumbling block with complexity is the limitations of

Human Short-term Memory

People don't

mind dealing with complexity

provided it is

Localised

The Starting Point

Requirements in Natural Language

- Large numbers of requirements overflow our short-term memory
- Ambiguity, and many other types of defects are not “visible” in sequential text
- Can’t grasp what system does as a “whole”
- How do we organize teams to work productively?

Formalization - Modelling our only hope

Formalization - Challenges

- Accuracy – How to preserve original intent?
- Validation – Understandable by stakeholders?
- Complexity – Avoid short term memory overflow
- Defects – To make defects “visible”
- Comprehending – To see as a “whole”
- Dividing up the work – Productive teams?

Formalization Strategy ???

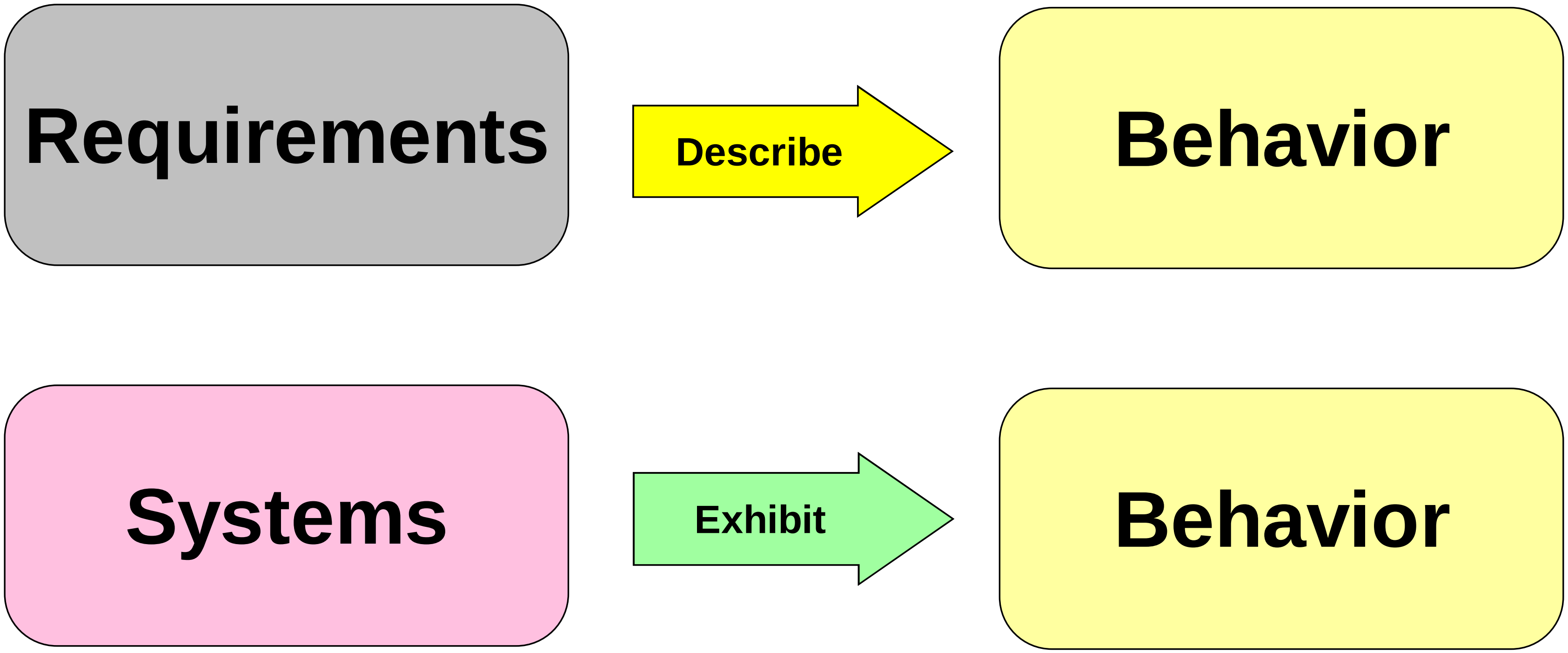
How do we do all this?

Strategy

Build system
<u>OUT OF</u>
its requirements

REPRESENTATION – is the key to doing this

Requirements ⇔ Systems ⇔ Behavior



The Link – Build systems <u>out of</u> requirements

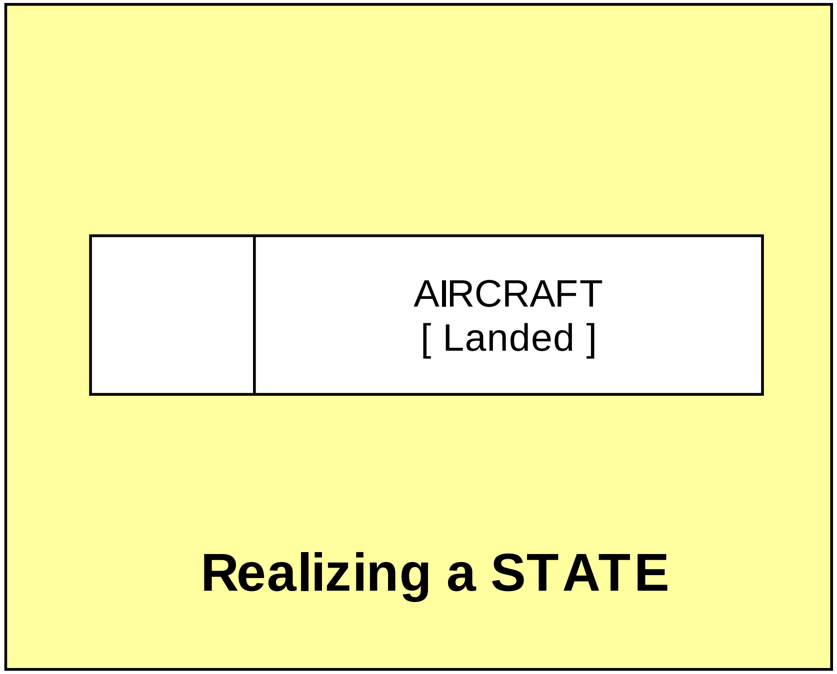
Two Types of Behavior

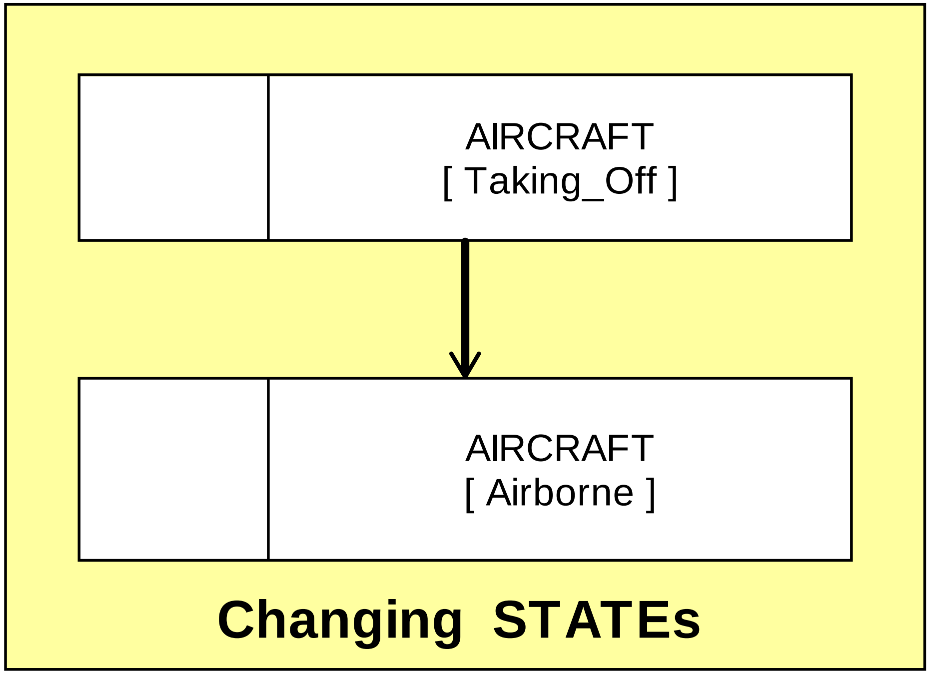
- Component behavior – component acting
- Network behavior – components interacting

The soccer player behavior versus soccer team behavior

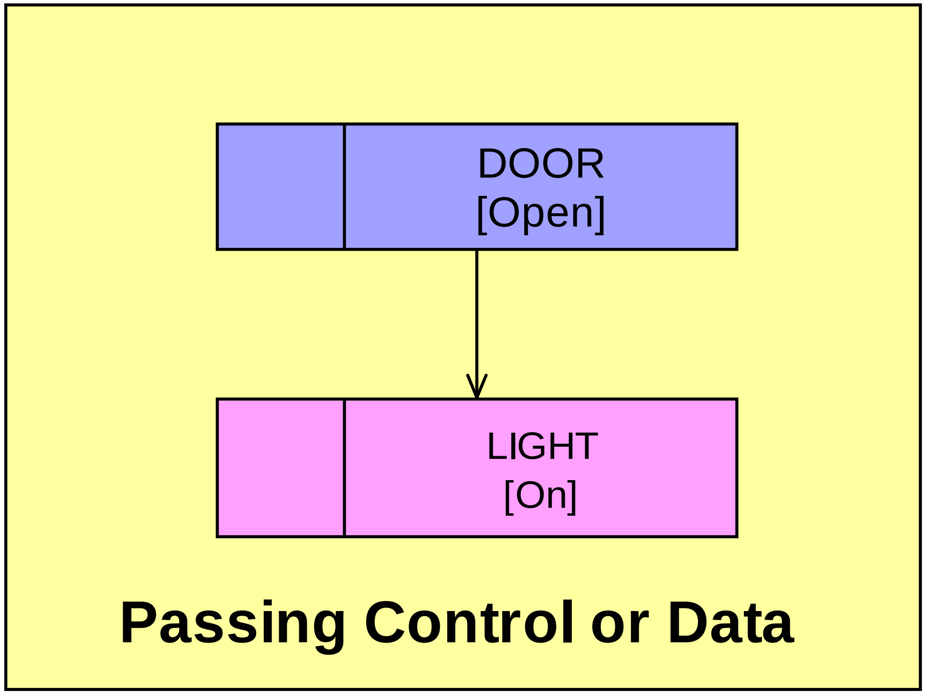
Formalization Using Behavior Trees

<u>Component behavior</u> – component acting





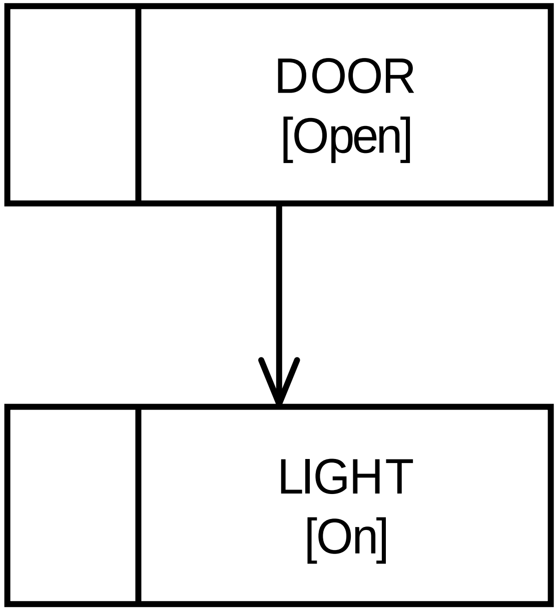
<u>Network behavior</u> – components interacting



Informal => Formal : by Translation

TEXTUALLY	GRAPHICALLY
Informal **Potentially Ambiguous**	**Formal Semantics** **Unambiguous**

"Whenever the door becomes open it causes the light to go on"



Behavior Tree

Components?

States?

Step 1.

Requirements Translation

Functional Requirement

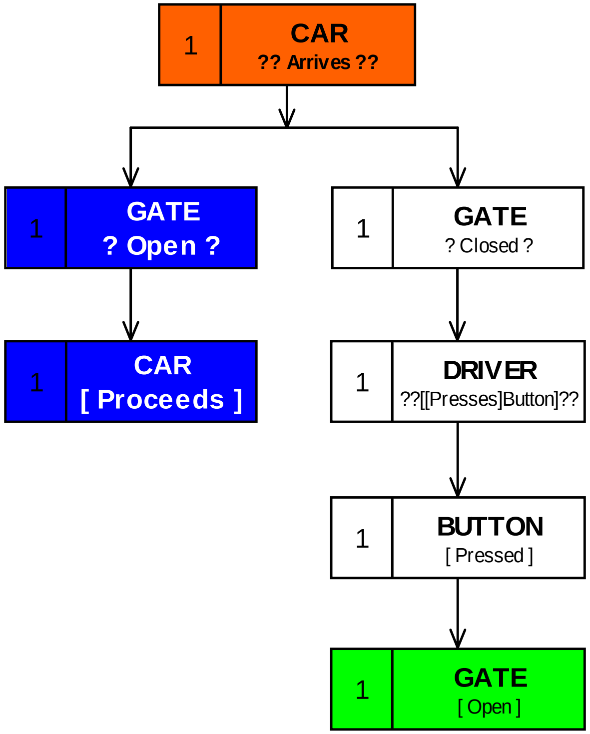
When a car arrives,
if the gate is open the car proceeds,
otherwise if the gate is closed, when the driver presses the button
it causes the gate to open

Requirements Translation

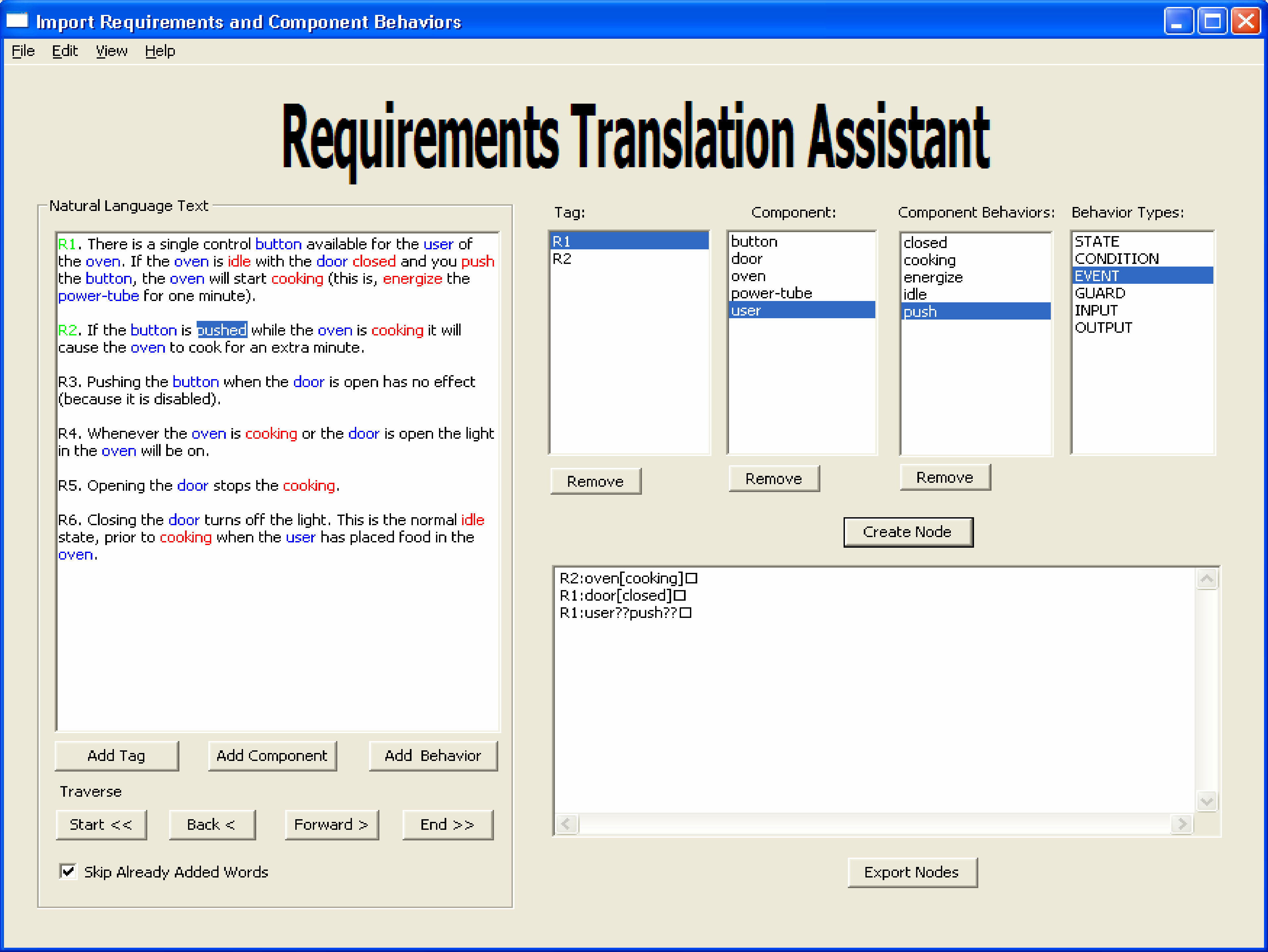
Functional Requirement

When a car is arrives,
if the gate is open the car proceeds,
otherwise if the gate is closed, when
the driver presses the button
it causes the gate to open

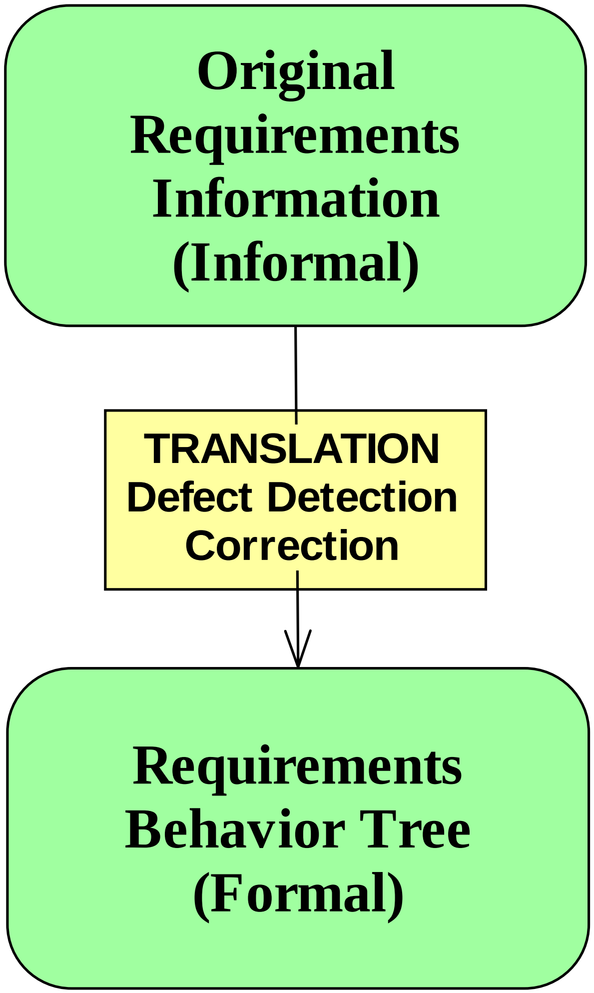
Behavior Tree



Informal => Formal


Import Requirements and Component Behaviors
File Edit View Help
Requirements Translation Assistant
Natural Language Text
R1. There is a single control button available for the user of the oven. If the oven is idle with the door closed and you push the button, the oven will start cooking (this is, energize the power-tube for one minute).
R2. If the button is pushed while the oven is cooking it will cause the oven to cook for an extra minute.
R3. Pushing the button when the door is open has no effect (because it is disabled).
R4. Whenever the oven is cooking or the door is open the light in the oven will be on.
R5. Opening the door stops the cooking.
R6. Closing the door turns off the light. This is the normal idle state, prior to cooking when the user has placed food in the oven.
Add Tag
Add Component
Add Behavior
Traverse
Start <<
Back <
Forward >
End >>
Skip Already Added Words
Tag:
R1
R2
Component:
button
door
oven
power-tube
user
Component Behaviors:
closed
cooking
energize
idle
push
Behavior Types:
STATE
CONDITION
EVENT
GUARD
INPUT
OUTPUT
Remove
Remove
Remove
Create Node
R2:oven[cooking]
R1:door[closed]
R1:user??push??
Export Nodes

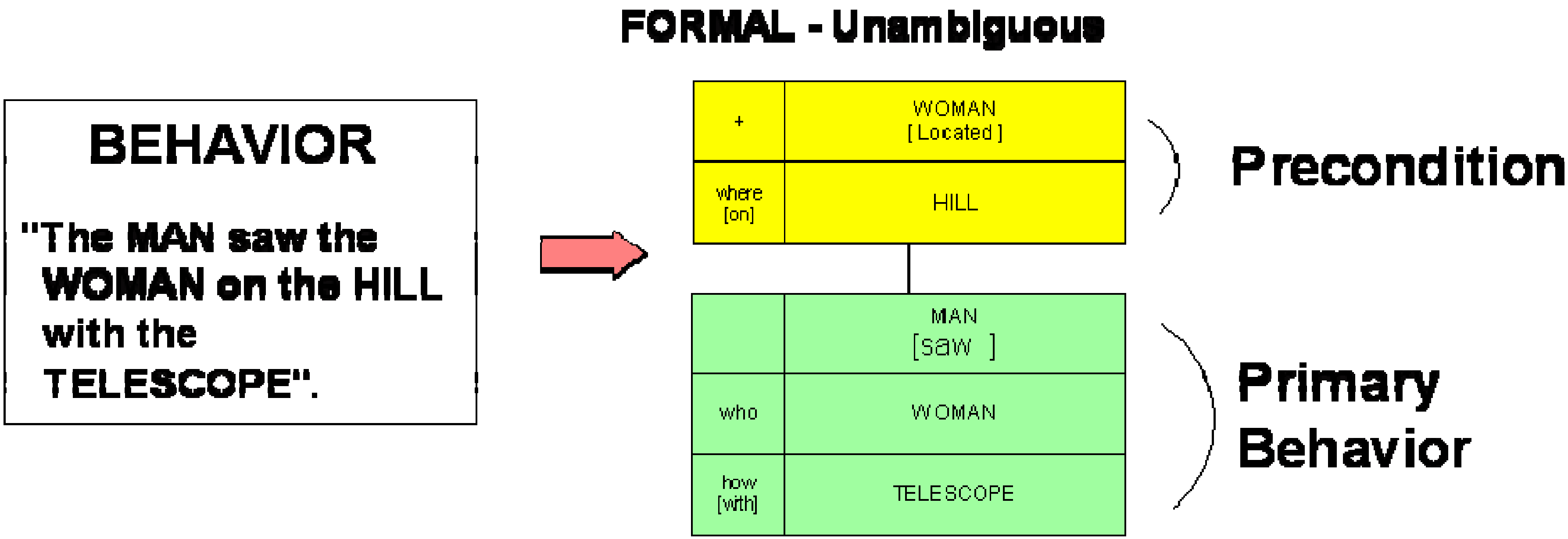
Requirements Translation



GOALS of Translation:

- To Preserve meaning
- To Clarify meaning
- Not to Add anything
- Not to leave out anything
- Not to modify anything
- Not to change vocabulary

Requirements Translation - Ambiguity



There are at least three interpretations of this behavior – each has a different formal representation – author must validate which one is intended.

When Lots of Requirements

FR-01

2.2.1 Report Satellite Health

Satellite health information is requested to aid in planning required satellite maintenance.

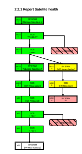
MG2) The SCS shall process each HR (satellite health request) command message received from the GCS.

MG2.1) An HR command message shall be the first message received after the initiation of each "Manage Satellites" transaction.

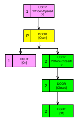
MG2.2) The SCS may receive a satellite health request message anytime during a "Manage Satellites" transaction. (Describes order during transaction.)

MG2.3) A satellite health request command message will only be accepted by the SCS during an active "Manage Satellites" transaction. (Describes condition under which an HR may be received.)

MG3) The SCS shall prepare and send an HA (satellite health acknowledgment) message to the GCS in response to an HR (satellite health request) command message.



Behavior Tree



Behavior Tree

FR-02

2.2.1 Report Satellite Health

Satellite health information is requested to aid in planning required satellite maintenance.

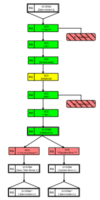
MG2) The SCS shall process each HR (satellite health request) command message received from the GCS.

MG2.1) An HR command message shall be the first message received after the initiation of each "Manage Satellites" transaction.

MG2.2) The SCS may receive a satellite health request message anytime during a "Manage Satellites" transaction. (Describes order during transaction.)

MG2.3) A satellite health request command message will only be accepted by the SCS during an active "Manage Satellites" transaction. (Describes condition under which an HR may be received.)

MG3) The SCS shall prepare and send an HA (satellite health acknowledgment) message to the GCS in response to an HR (satellite health request) command message.



Behavior Tree

● ● ●

FR-M

2.2.1 Report Satellite Health

Satellite health information is requested to aid in planning required satellite maintenance.

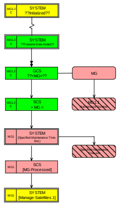
MG2) The SCS shall process each HR (satellite health request) command message received from the GCS.

MG2.1) An HR command message shall be the first message received after the initiation of each "Manage Satellites" transaction.

MG2.2) The SCS may receive a satellite health request message anytime during a "Manage Satellites" transaction. (Describes order during transaction.)

MG2.3) A satellite health request command message will only be accepted by the SCS during an active "Manage Satellites" transaction. (Describes condition under which an HR may be received.)

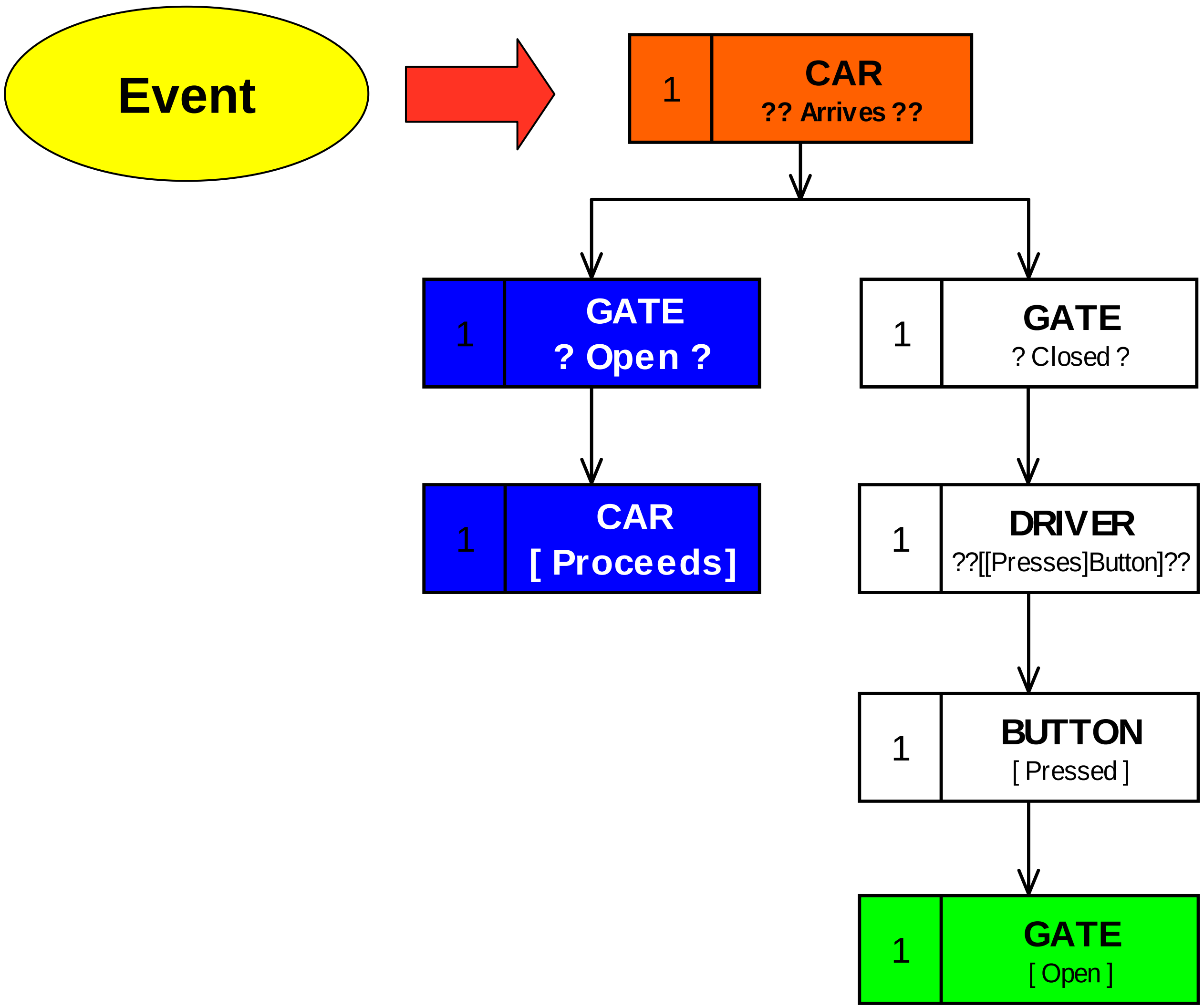
MG3) The SCS shall prepare and send an HA (satellite health acknowledgment) message to the GCS in response to an HR (satellite health request) command message.



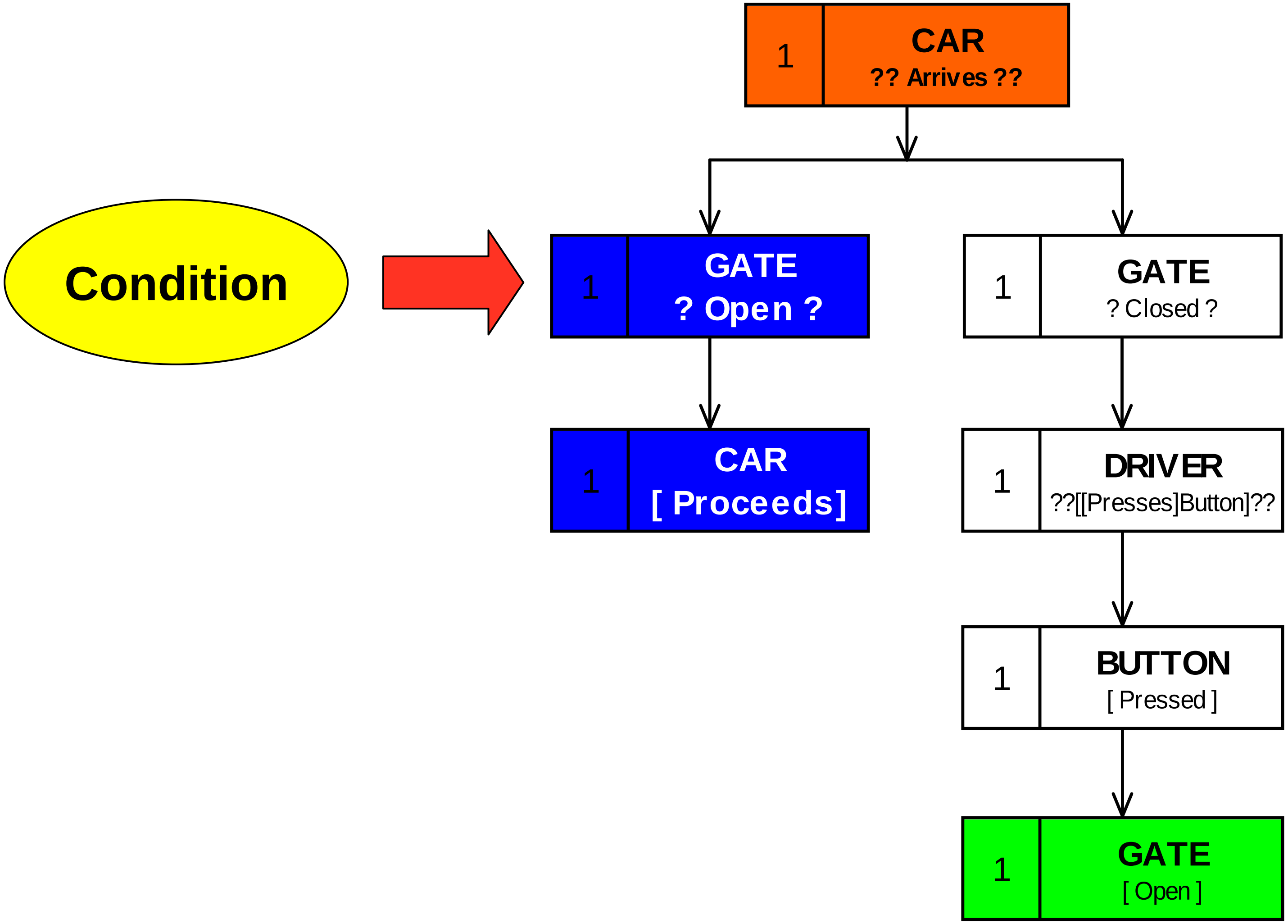
Behavior Tree

Behavior Tree Notation

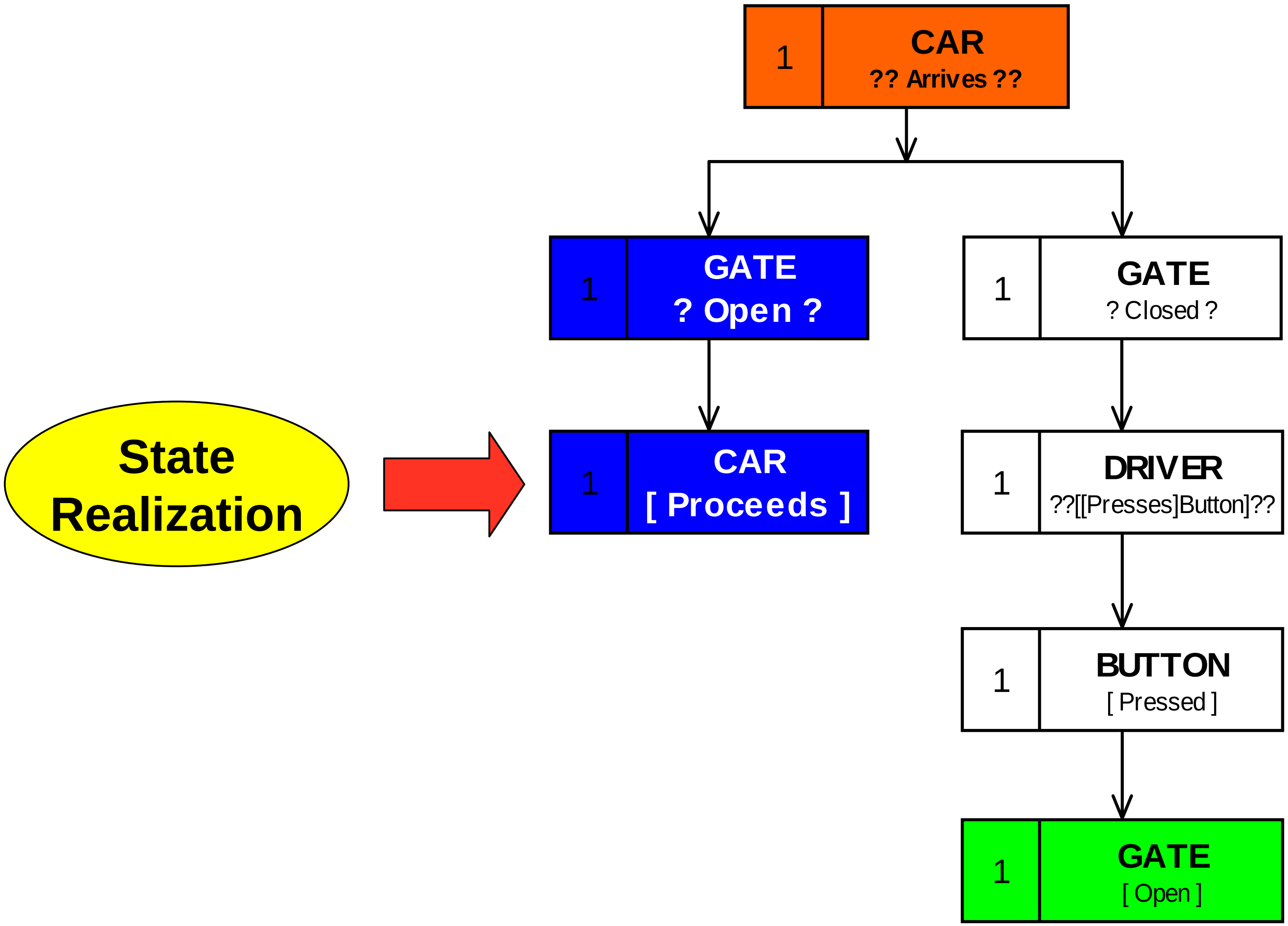
Behavior Tree Notation



Behavior Tree Notation



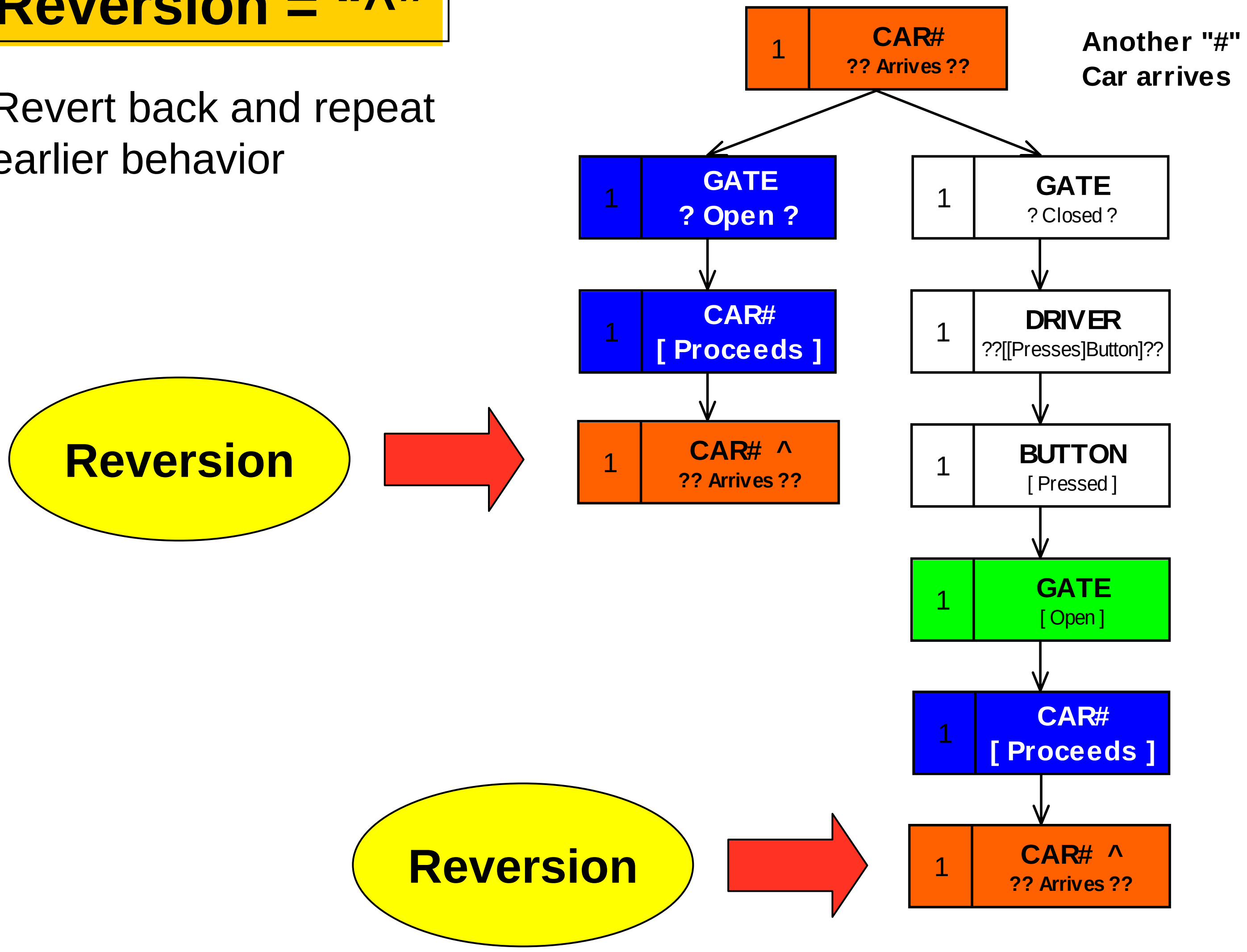
Behavior Tree Notation



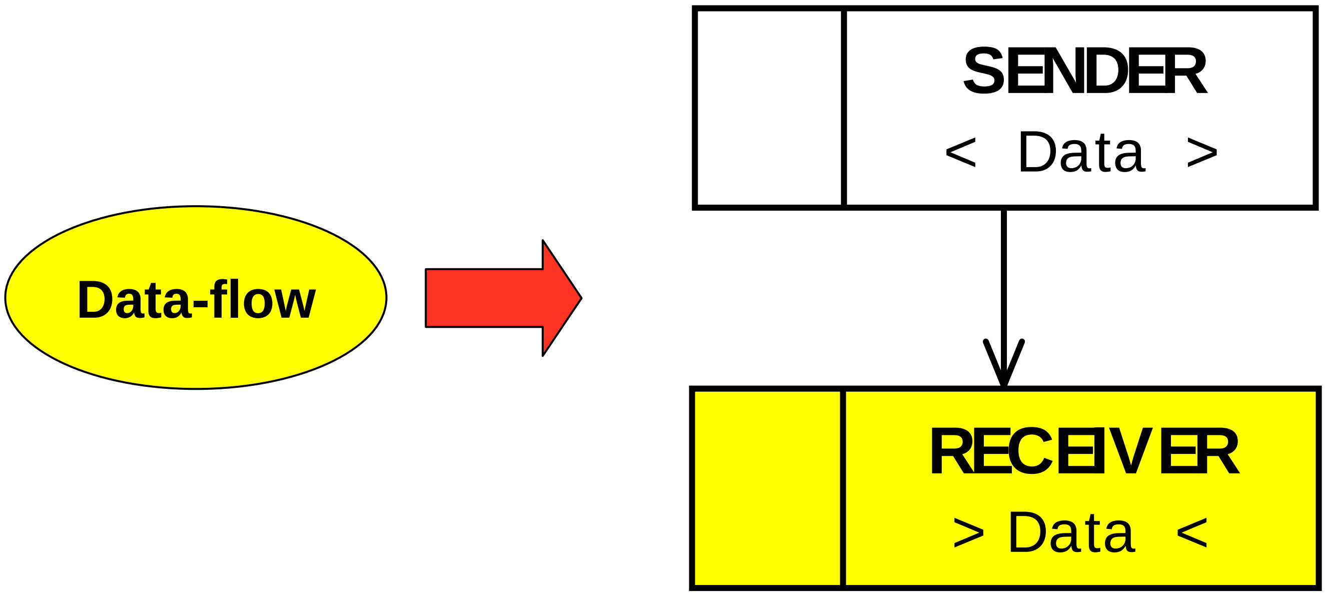
Behavior Tree Notation

Reversion = “^”

Revert back and repeat earlier behavior



Behavior Tree Notation



Behavior Tree Notation

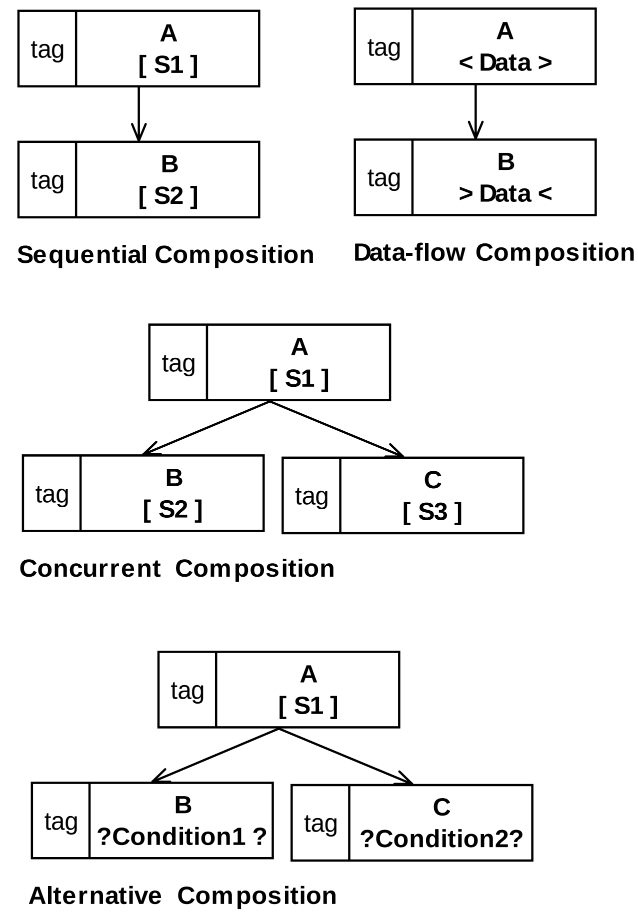
1. The telephone is on ...	TELEPHONE [On]	**State Realization**
2. When the telephone rings ...	TELEPHONE ?? Rings ??	**Event**
3. If the telephone is on ...	TELEPHONE ? On ?	**Condition**
4. The telephone sends message ...	TELEPHONE < Message >	**Data-out**
5. The telephone receives message...	TELEPHONE > Message <	**Data-in**
6. Telephone number assigned ...	TELEPHONE [Number := 55040]	**Attribute set**

Core Elements

Behavior Tree Notation

Component-State	Label	Semantics
tag · Component [State]	**State - Realization**	Component realizes State then passes control to its output.
tag · Component [Attribute := Value]	**Attribute-Assignment**	Component assigns a Value to one of its Attributes. Tag traces to requirement
tag · Component ? Condition ?	**Conditional - flow of control**	Component passes control to its output only if Condition is TRUE.
tag · Component ?? Event ??	**Event - flow of control**	Component only passes control when and if Event happens after reaching this component-state.
tag · Component ??? State ???	**Guard - flow of control**	Component passes control when State is realized an another thread or has happened prior to reaching this component-state.
tag · Component < Data_Out >	**Data Output State**	Component outputs Data_Out to the receiving component connected to its output
tag · COMPONENT > Data_Out <	**Data Input State**	Component inputs Data_Out from the sending component connected to its input
tag · System [State]	**System-State Realization**	System component realizes State then passes control to its output.

Composition Examples



Core Elements

Behavior Tree Notation

- The BT notation captures in a simple tree-like form of composed component-states what usually needs to be expressed in a mix of other representations.

- The language elegantly captures three types of inter-process communication: *shared variable*, *synchronization* and *message passing*.

- BTs have been given a formal semantics based on a low-level process algebra, BTPA.

- BTs can be used to support model-checking simulation and code generation.

Where to Next?

Step 2.

Requirements Integration

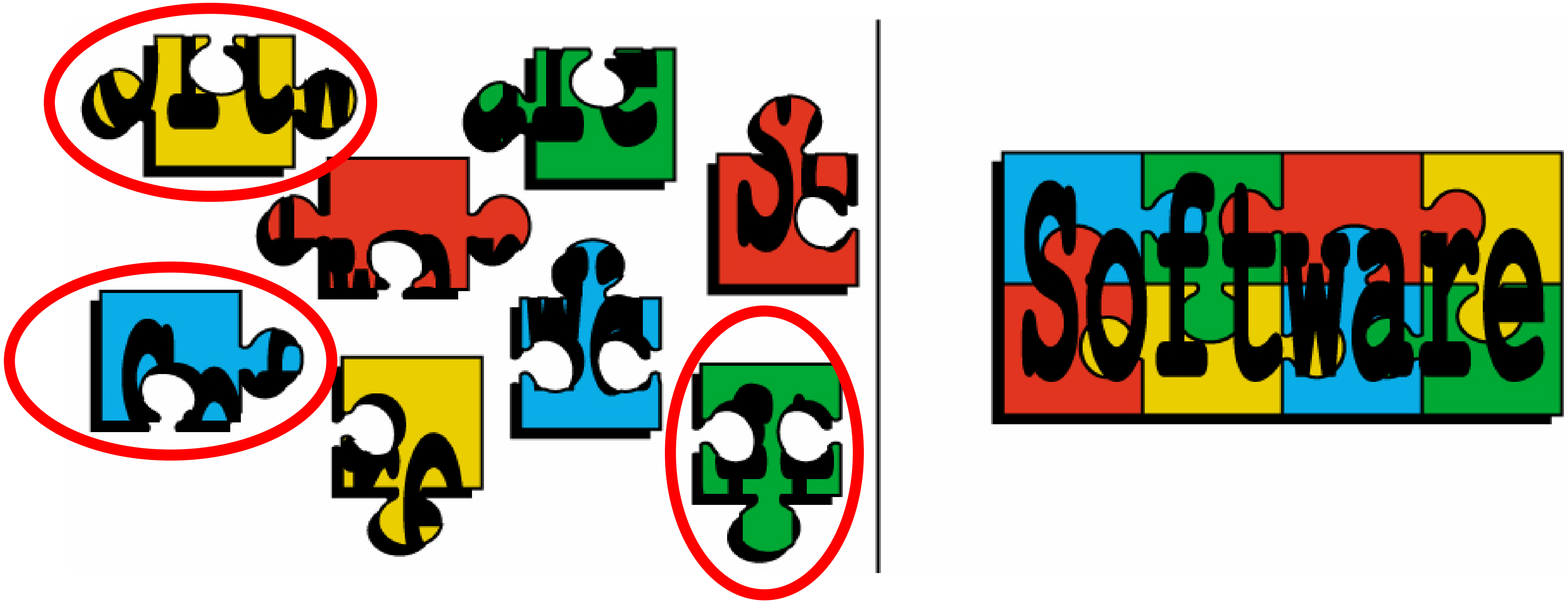
Putting the pieces together

Observation – Jigsaw Puzzles

The pieces of a jigsaw puzzle (and model toy kits) have the interesting (genetic) property that:

they contain <u>enough information</u> to allow the pieces to all be assembled one at a time.

Creating an Integrated View



- Order is not important BUT position where placed is
- Information about "f" is spread across THREE pieces
- Only see there are missing pieces when integrate pieces
- Only see some pieces don't fit when integrate pieces

Same IDEAS apply with requirements

Creating an Integrated View

A set of requirements also:

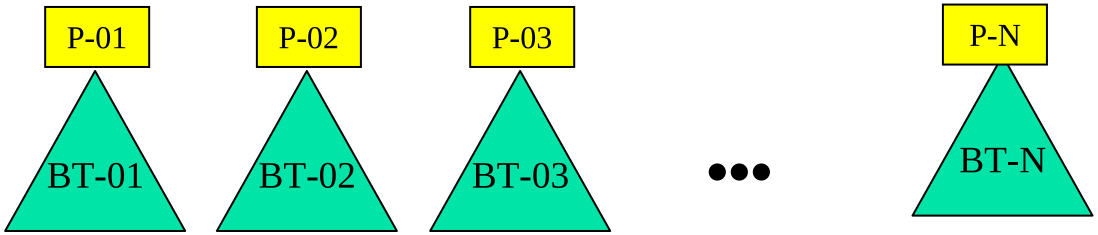
contain enough information to allow them to all be assembled

into an integrated view which becomes a precursor to the system design

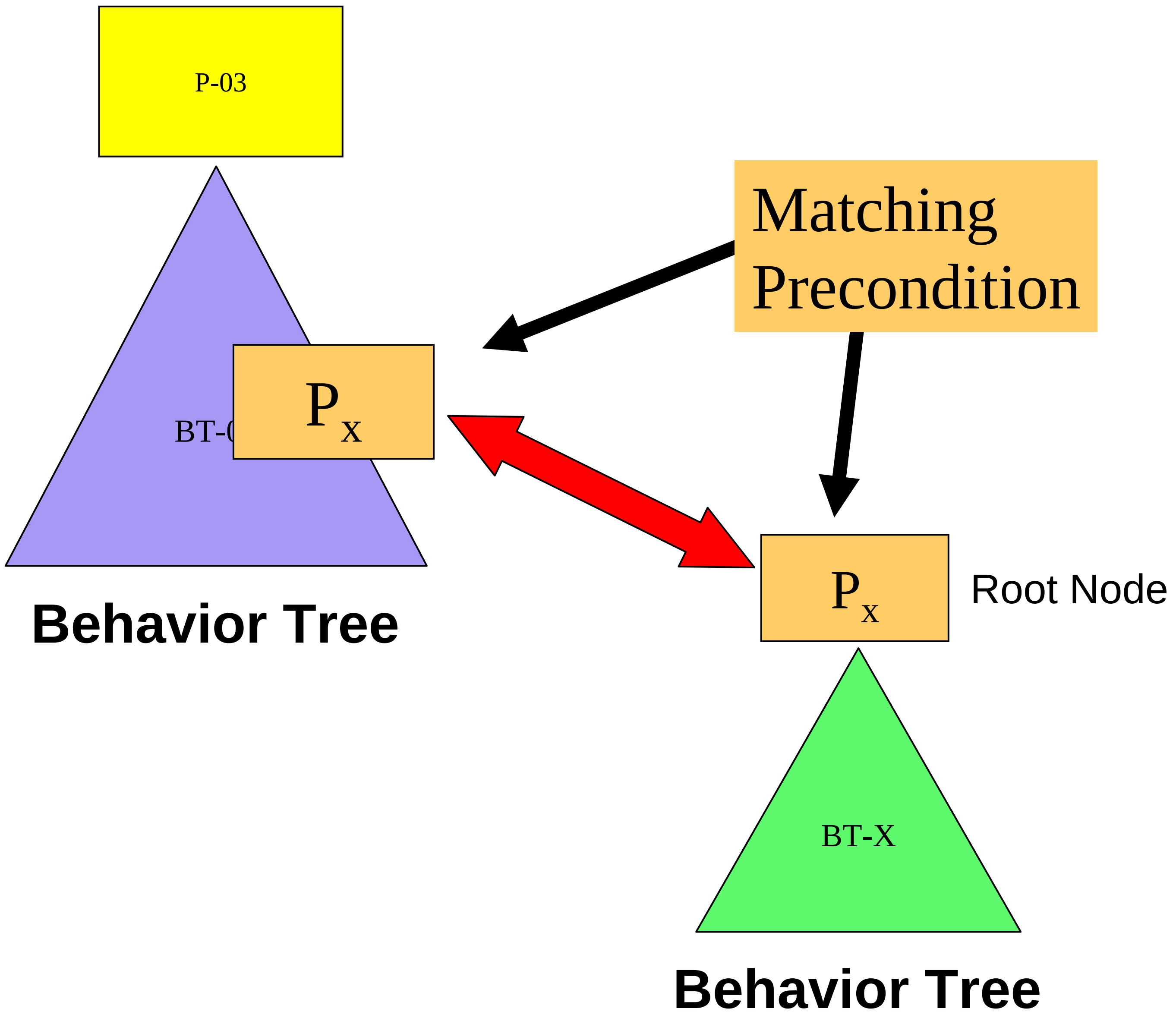
Proviso - you need to use the right representation

Enabler - Precondition Axiom

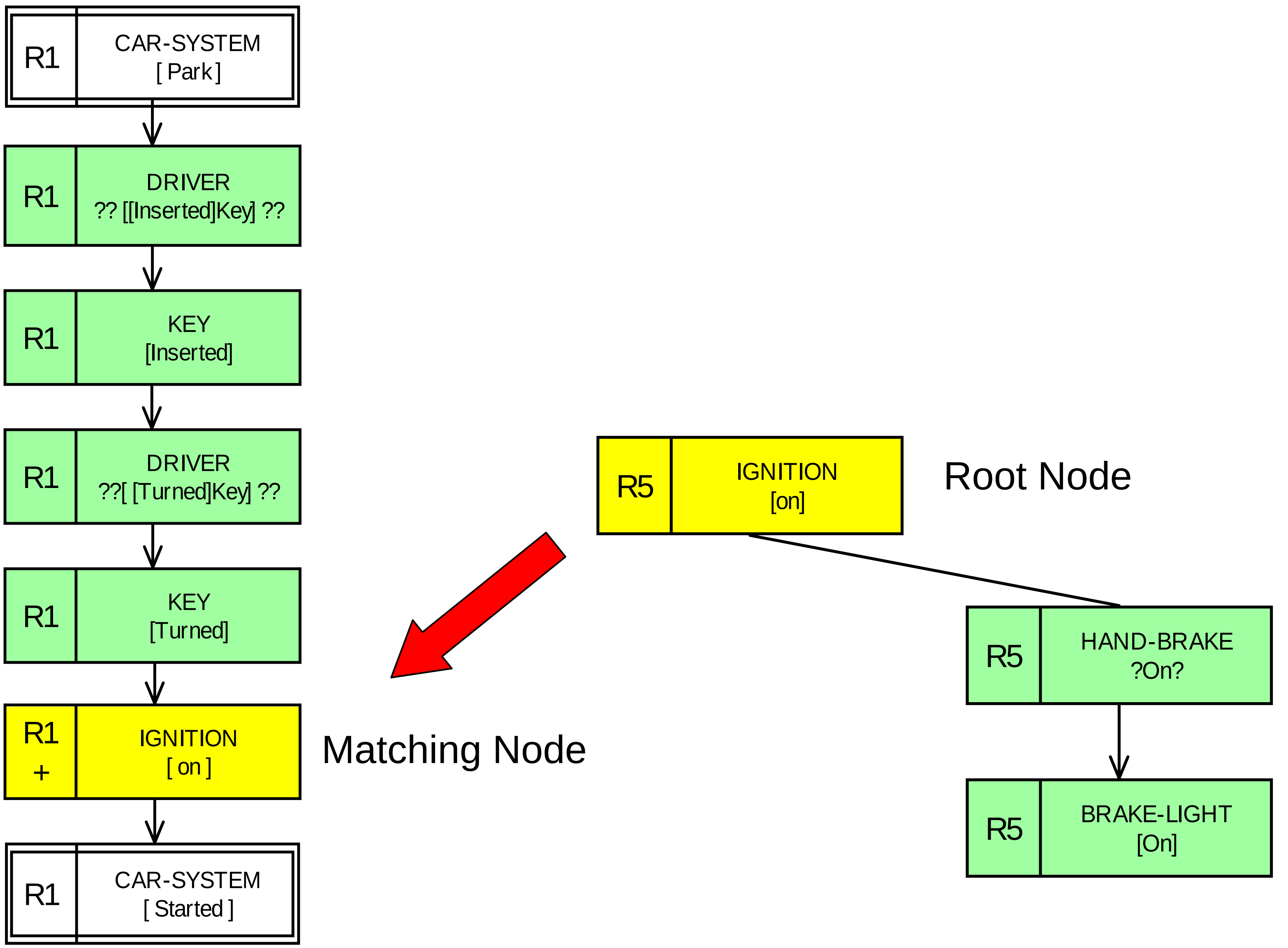
Each and every functional requirement expressed as a behavior tree BT_i has associated with it a precondition P_i that needs to be satisfied in order for the behavior encapsulated within it to be exhibitable



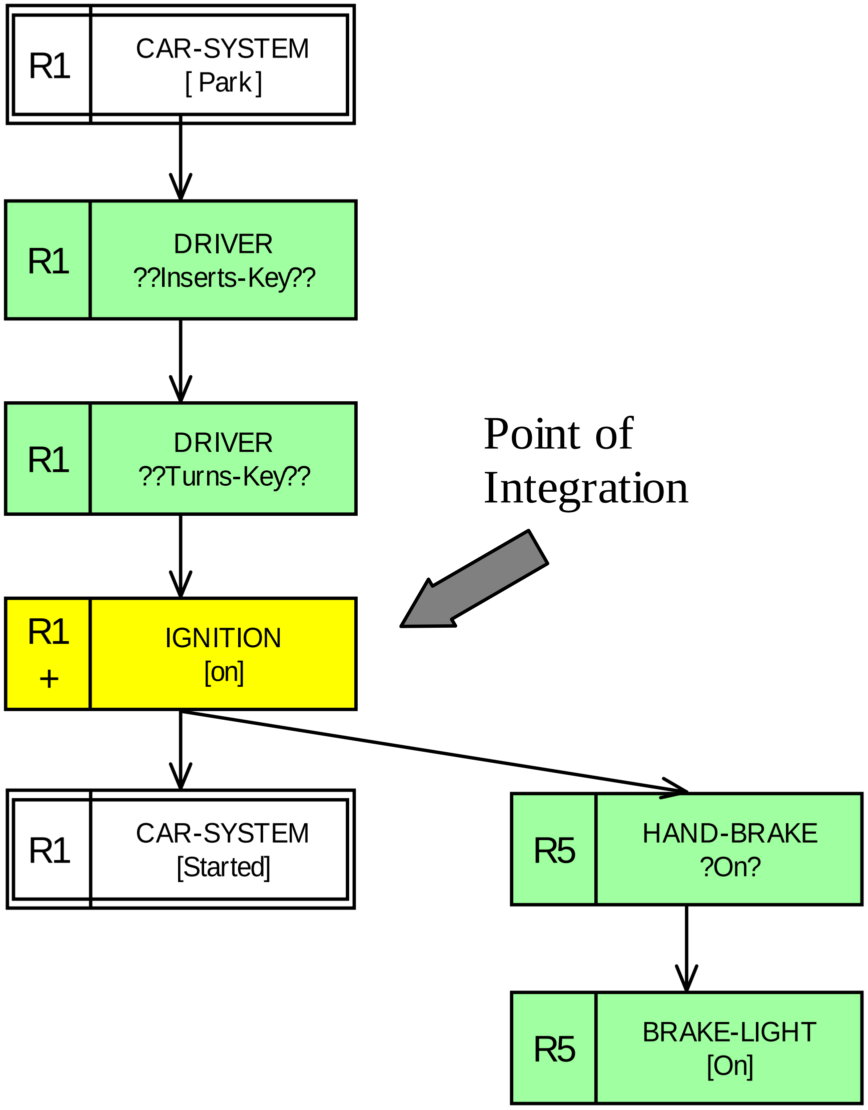
Enabler - Interaction Axiom



Requirements Integration



Requirements Integration



Requirements Integration

Find where root node of one tree occurs in another tree – **JOIN** at that point.

How Does This Help ?

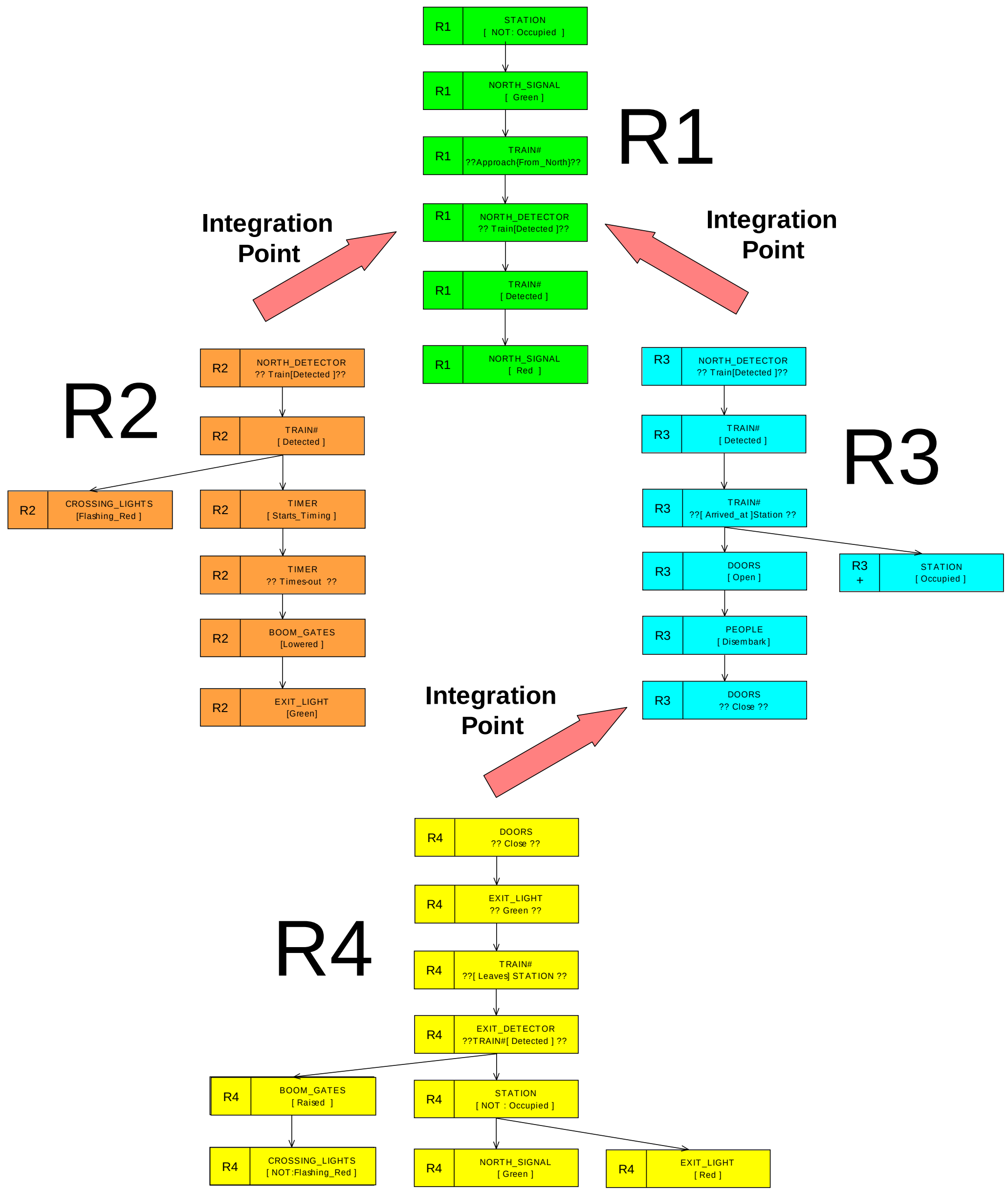
If we have a large number of requirements each can be INTEGRATED into a <u>behavior tree</u>

ONE AT A TIME

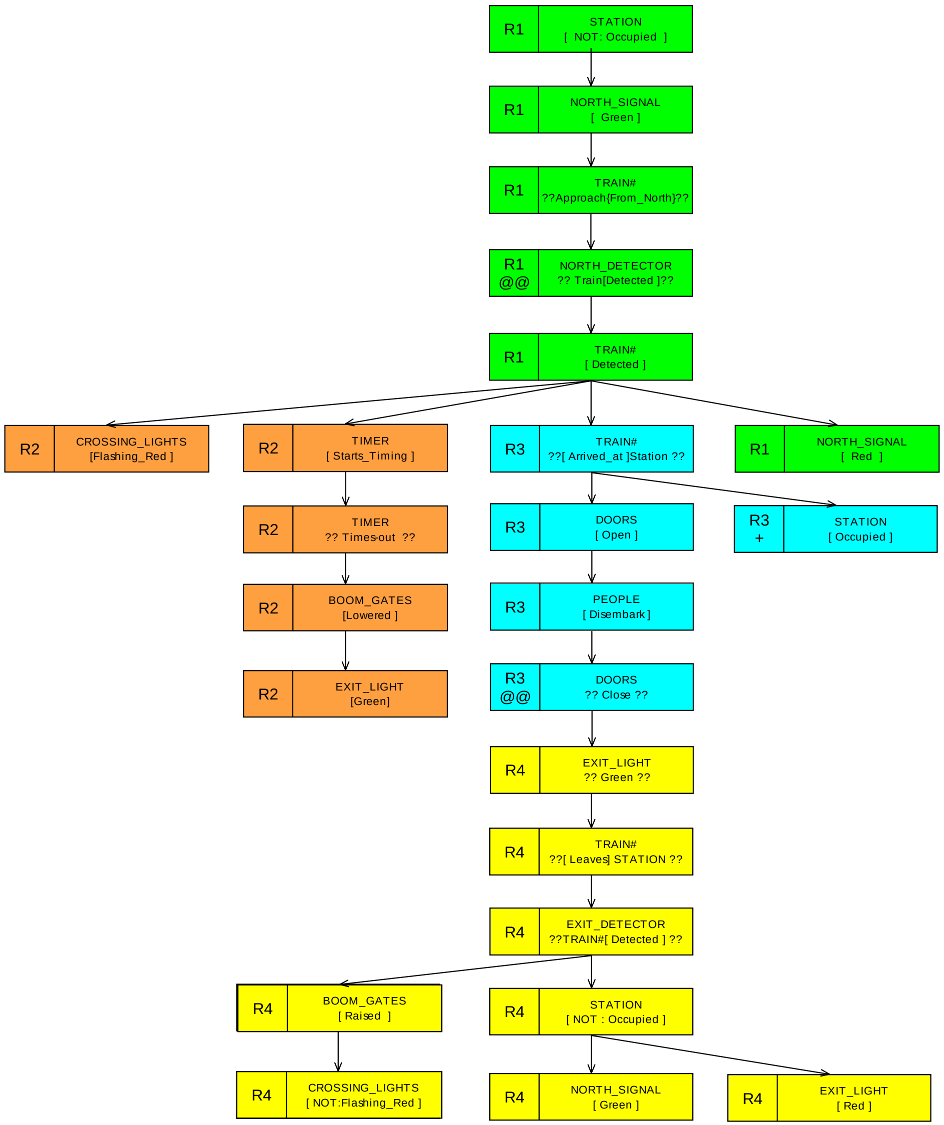
It allows us to deal with complexity

Creating an Integrated Behavioral View From Requirements

Requirements Integration



Requirements Integration



Build system out of its requirements

An Example

Example – Train Station System

TRAIN-STATION PROBLEM (Sherwood Station)

Develop a system to model the behavior of a Train-Station. You need to model a train entering the station from the north and then leaving the station to the south. A crossing with boom gates and flashing red lights is located just south of the station. There is a signal to the north of the station that only allows a train to enter when the station is not occupied, that is, when the north signal is green. There is also an exit signal light that ensures the train can only leave the station when the boom gates are down. There is also a north detector that can detect the train approaching the station region from the north. And, there is an exit detector that detects when a train leaves to the south.

1. Initially the station is not occupied. The north signal turns green whenever the station is not occupied. Whenever the north signal is green a train may approach from the north. When approaching from the north a train is detected, by the north detector, which causes the north signal to turn red.

2. When the north detector detects a train it causes the crossing lights to start flashing red. At the same time, a timer starts timing and when it times out it causes the boom gates to be lowered after which the exit light turns green.

3. After the train is detected the north detector, it subsequently arrives at the station, the doors open, the people disembark, and then the doors close.

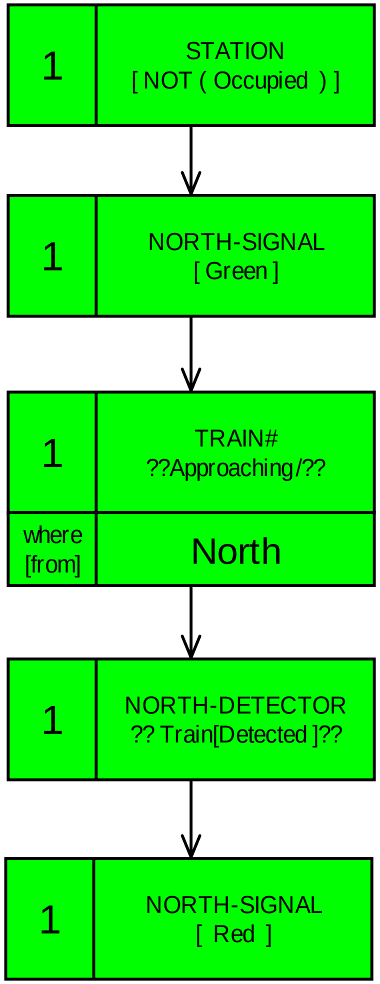
4. After the doors close the train may leave the station only when and if the exit light is green. When the train leaves the station, heading south, it is detected by the exit detector which means the station is again not occupied. This causes the north signal to turn green and the exit light to turn red. When the exit detector detects the train leaving, it also causes the boom gates to be raised and then the crossing lights to stop flashing red.

For the purposes of the exercise ignore trains approaching the station from the south. This additional requirement can be integrated later as a separate exercise. Also ignore situations where the train does not stop at the station - this too requires some refinements to the design.

Step 1.

Requirements Translation

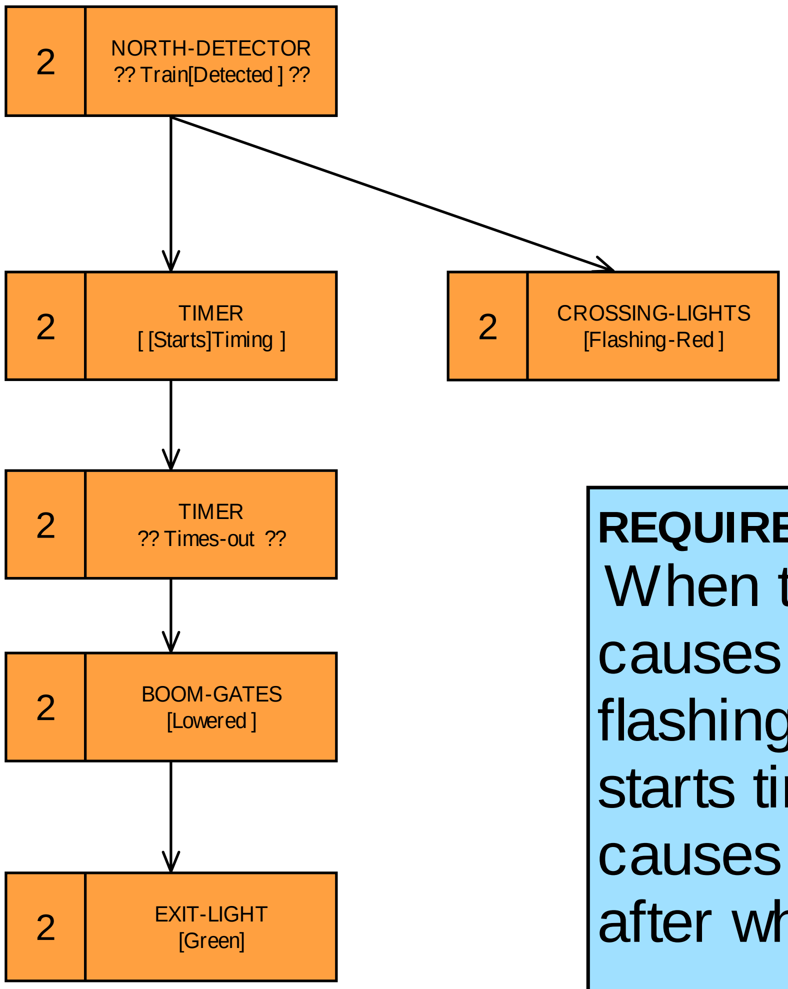
R1 – Translated Behavior Tree



REQUIREMENT-R1

Initially the station is not occupied. The north signal turns green whenever the station is not occupied. Whenever the north signal is green a train may approach from the north. When approaching from the north, a train is detected by the north detector, which causes the north signal to turn red.

R2 – Translated Behavior Tree

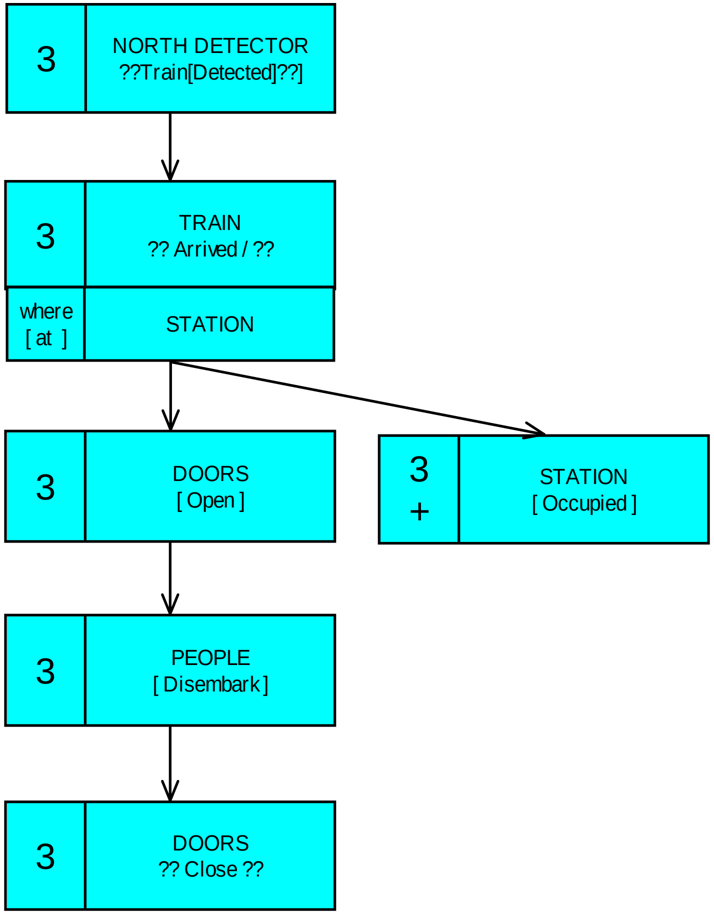


REQUIREMENT-R2

When the north detector detects a train it causes the crossing lights to start flashing red. At the same time a timer starts timing and when it times out, it causes the boom gates to be lowered, after which the exit light turns green.

R3 – Translated Behavior Tree

REQUIREMENT-R3
After the train is detected by the north detector, it subsequently arrives at the station, the doors open, the people disembark, and then the doors close.

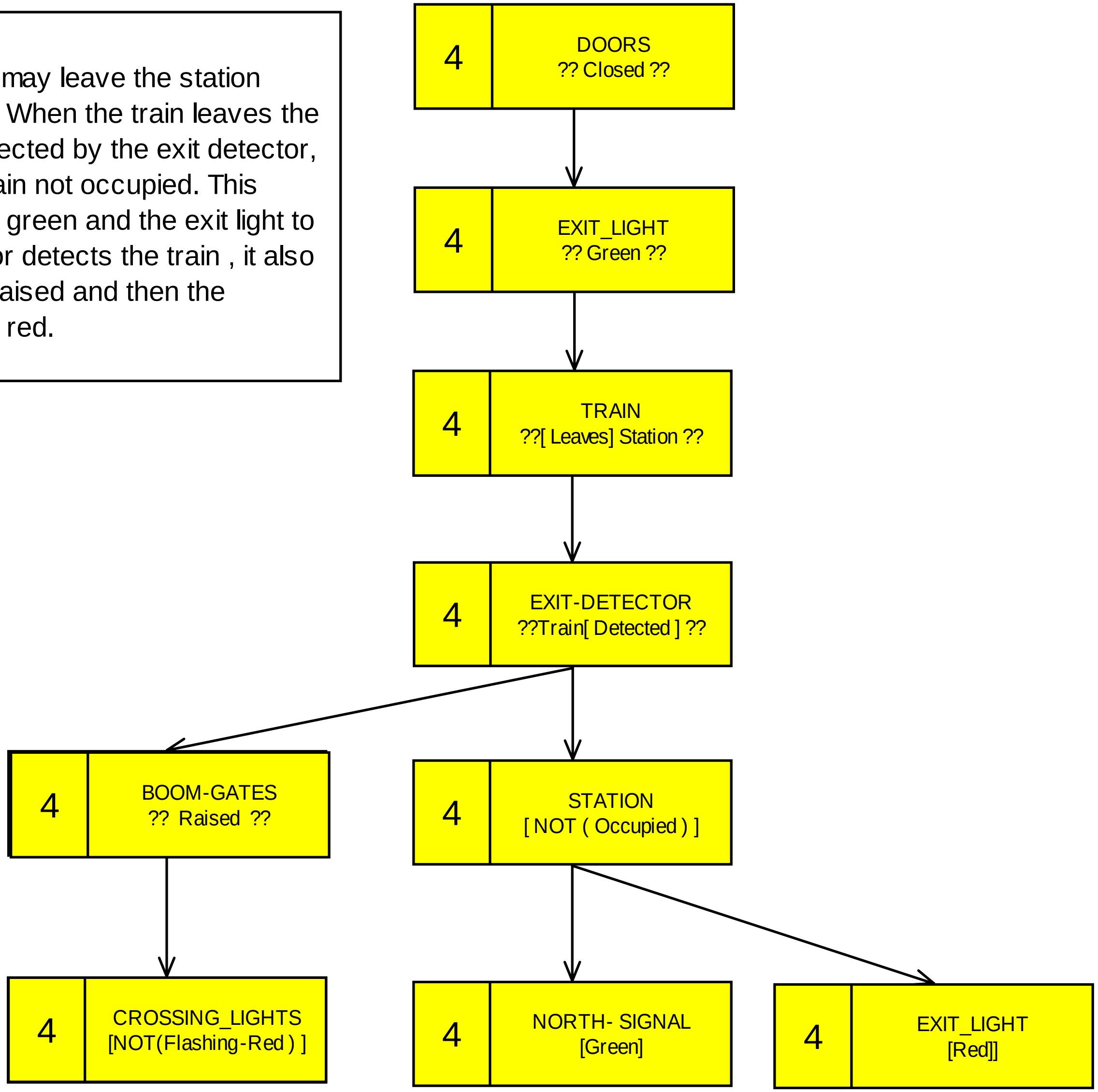


Implied (+)

R4 – Translated Behavior Tree

REQUIREMENT-R4

After the doors close the train may leave the station provided the exit light is green. When the train leaves the station, heading south, it is detected by the exit detector, which means the station is again not occupied. This causes the north signal to turn green and the exit light to turn red. When the exit detector detects the train , it also causes the boom gates to be raised and then the crossing lights to stop flashing red.

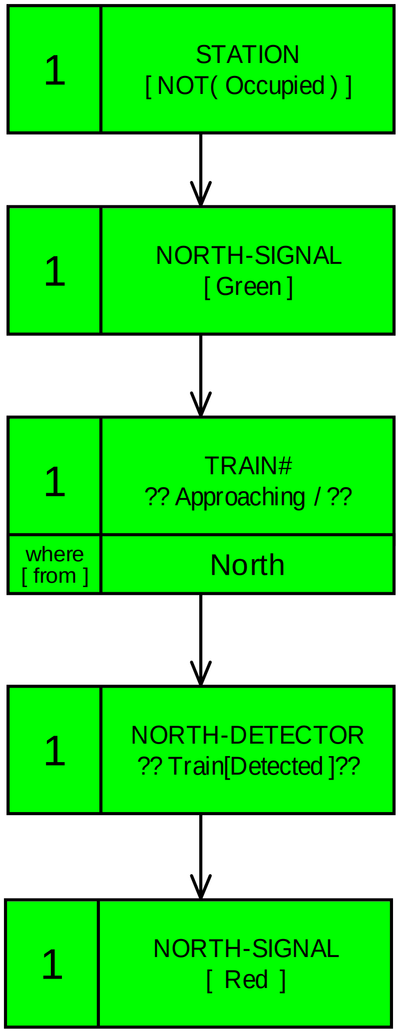


Step 2.

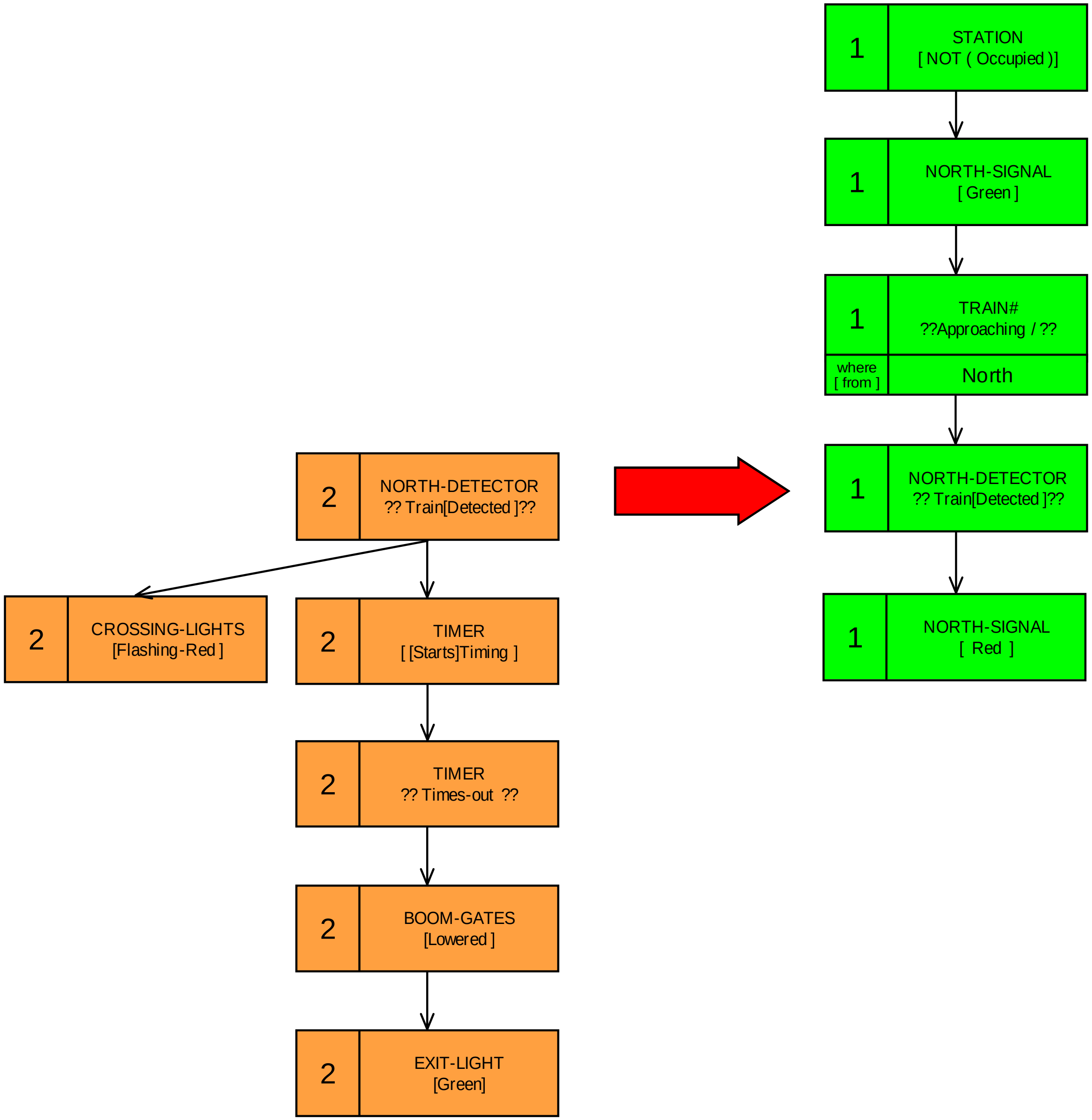
Requirements Integration

Putting the pieces together

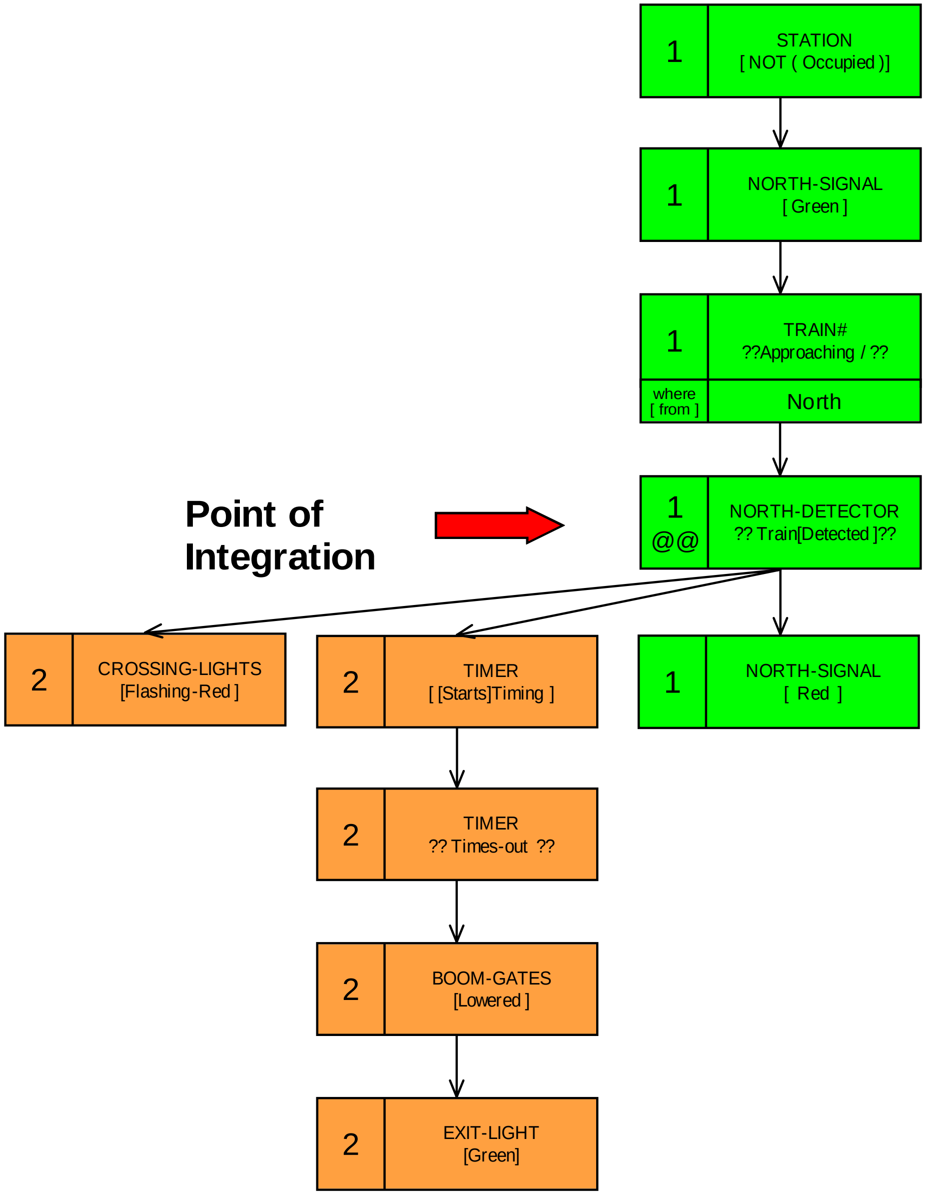
Integration – Base Case



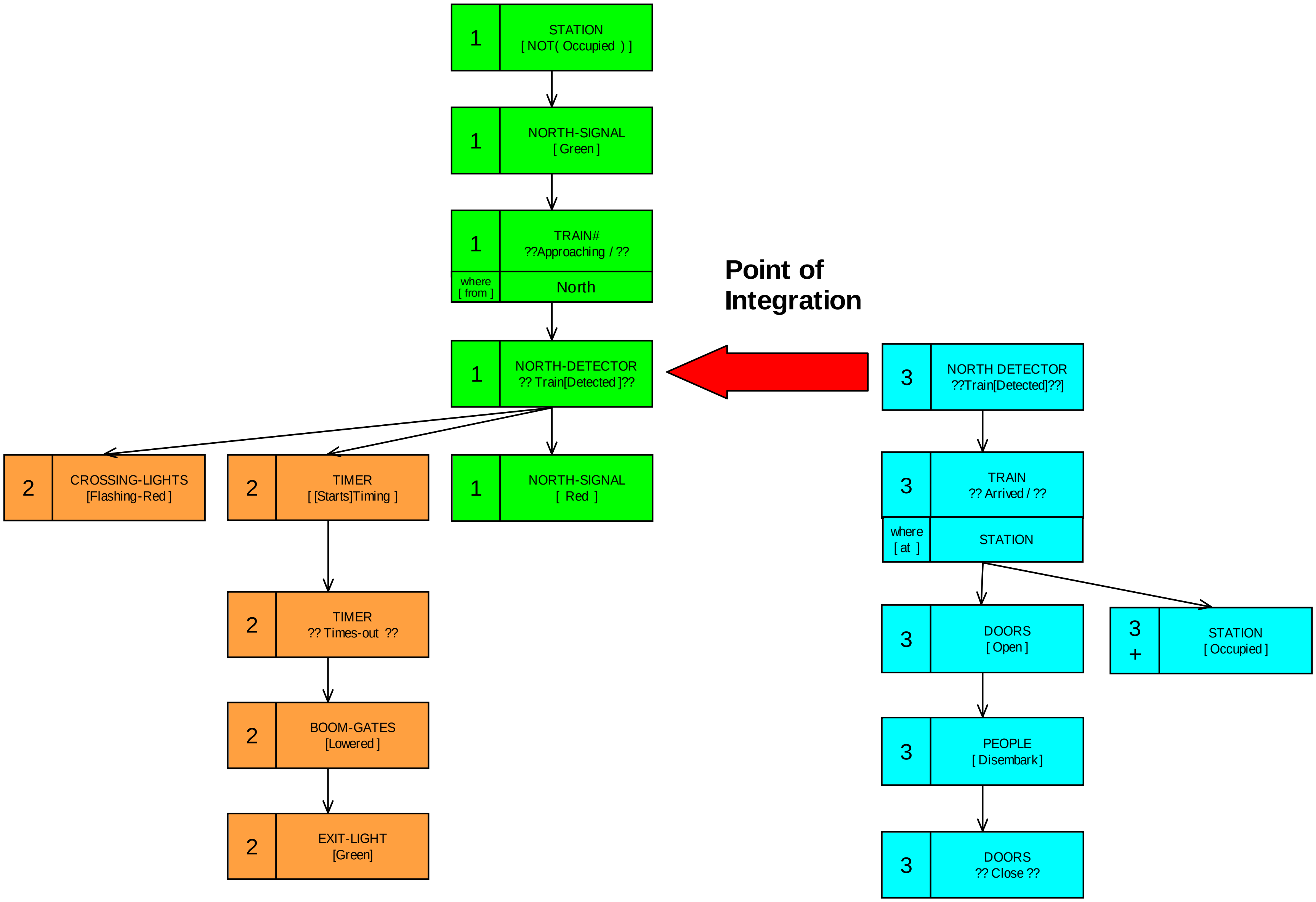
Integration of R2 with R1



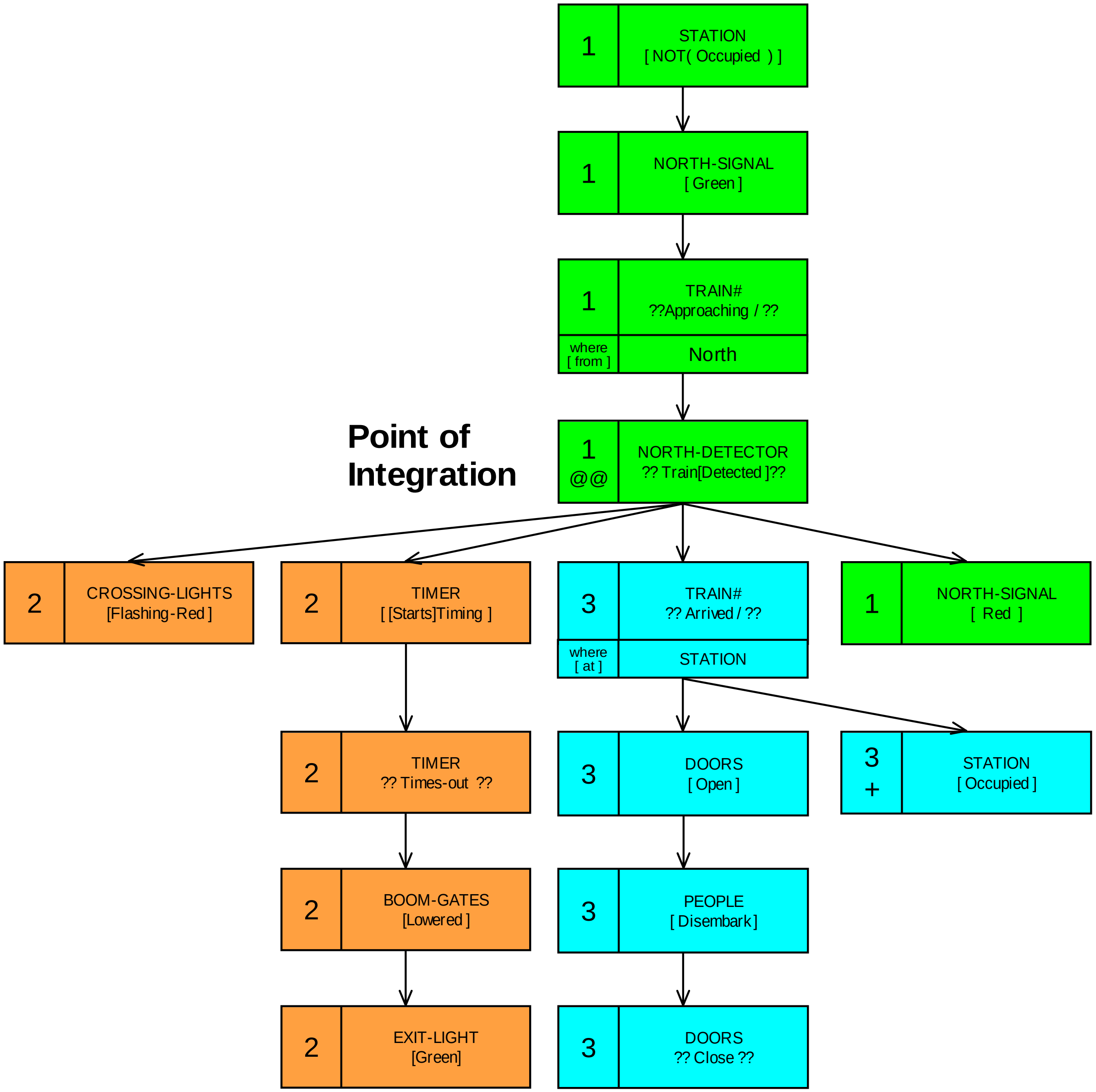
Integration of R2 with R1



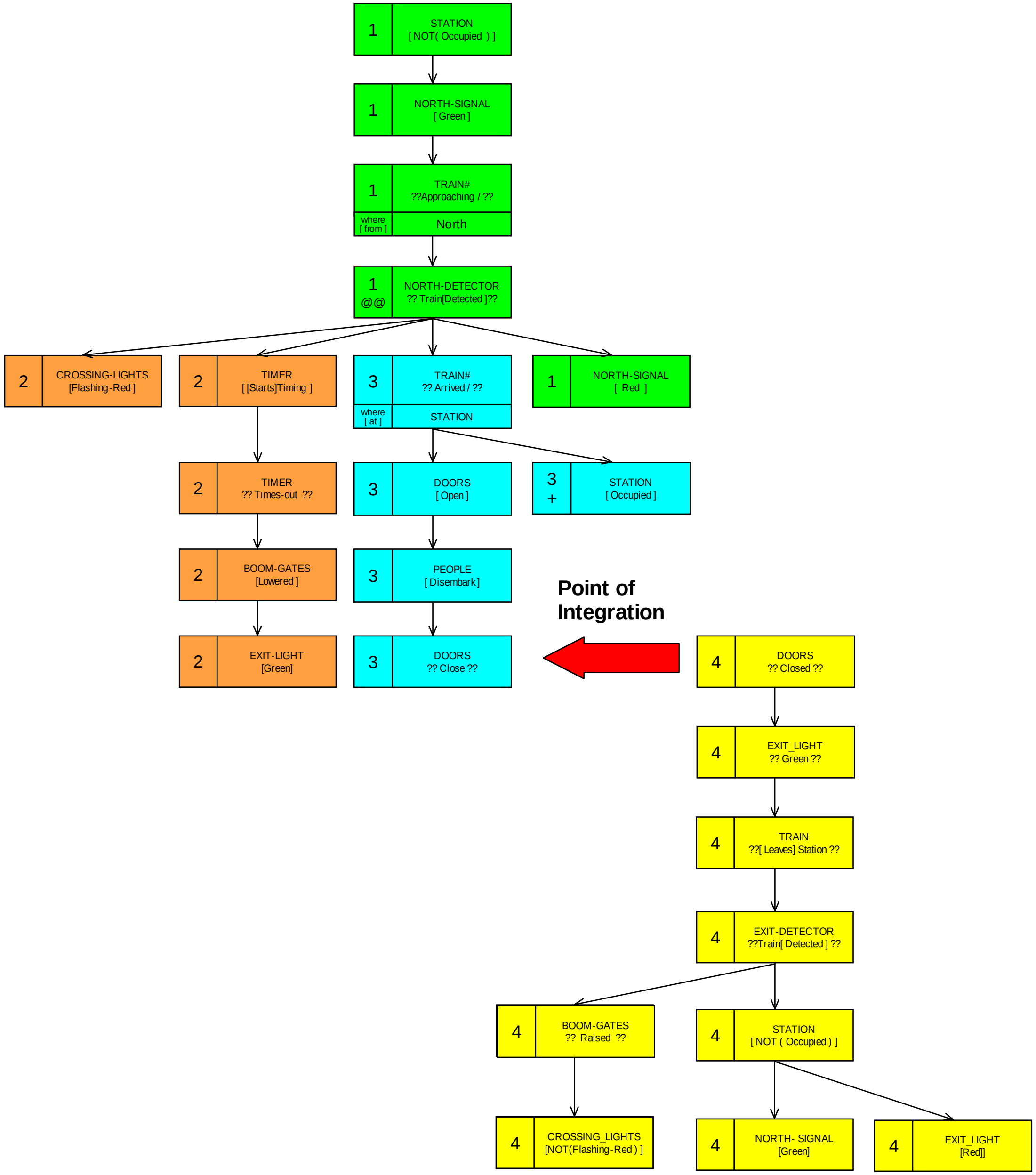
Integration of R3 into IBT



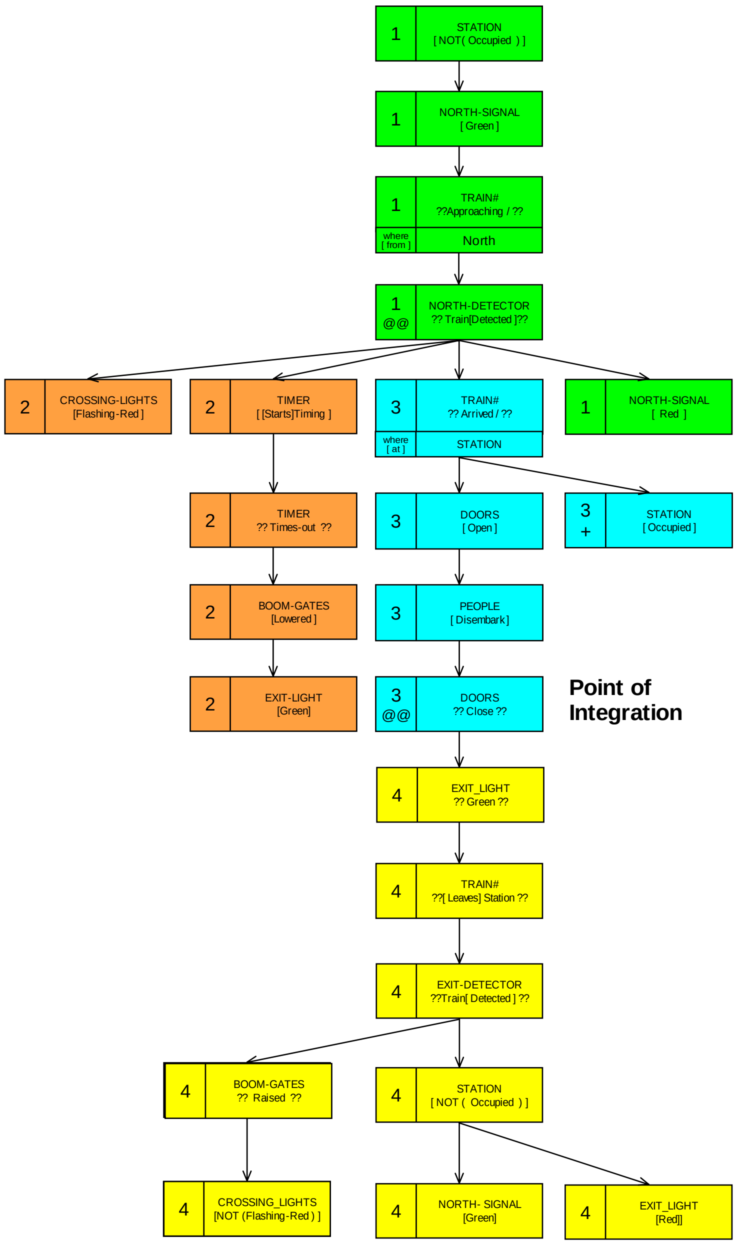
Integration of R3 into IBT



Integration of R4 into IBT



Integration of R4 into IBT



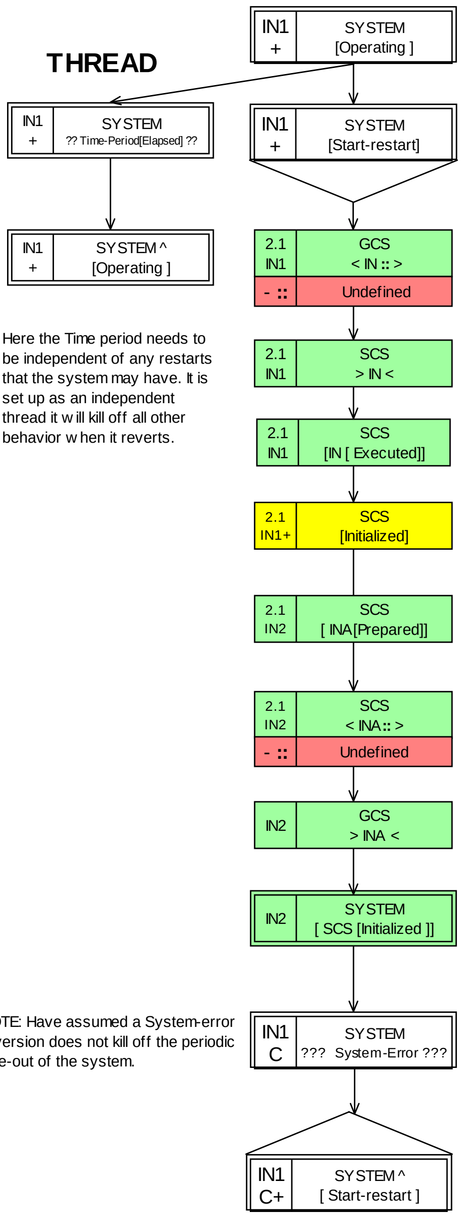
Integrated Behavior Tree

Larger system – 40 pages

Second Example

Satellite Control System
Case Study

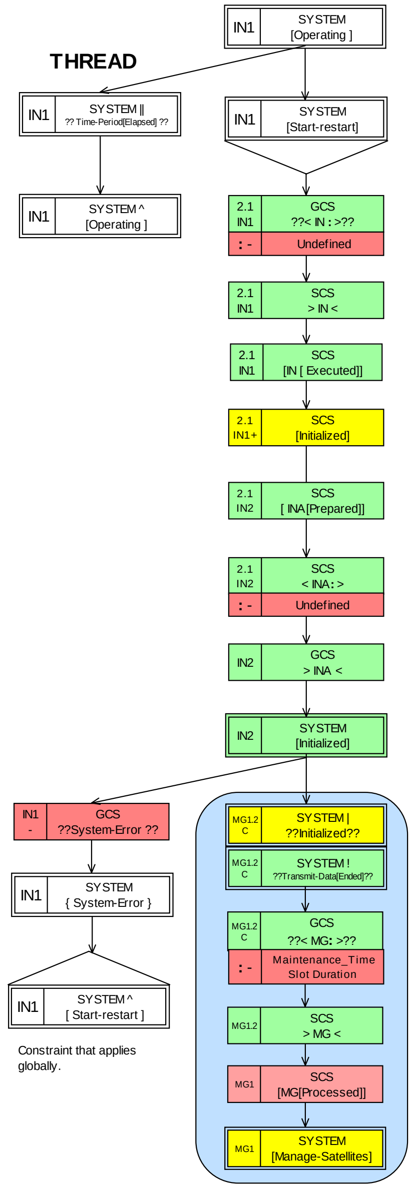
INT-01 (Base Case 2.1 Initialization)



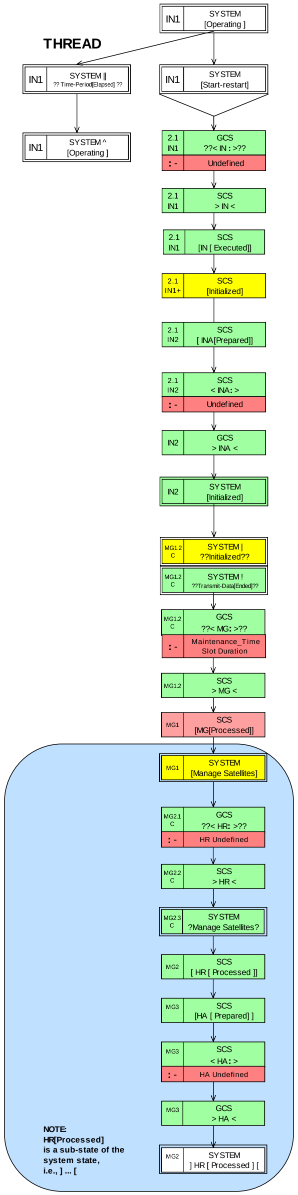
Here the Time period needs to be independent of any restarts that the system may have. It is set up as an independent thread it will kill off all other behavior when it reverts.

NOTE: Have assumed a System-error reversion does not kill off the periodic time-out of the system.

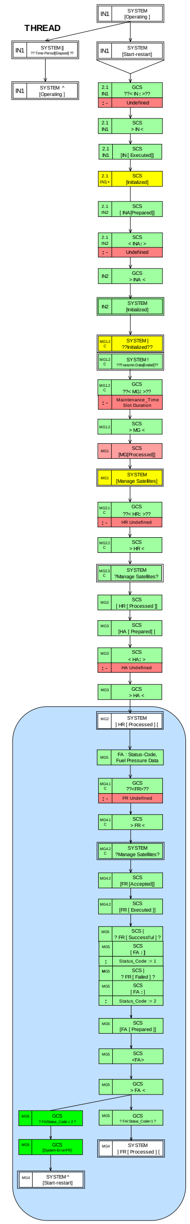
INT-02 (Integrating 2.2 Manage Satellites)



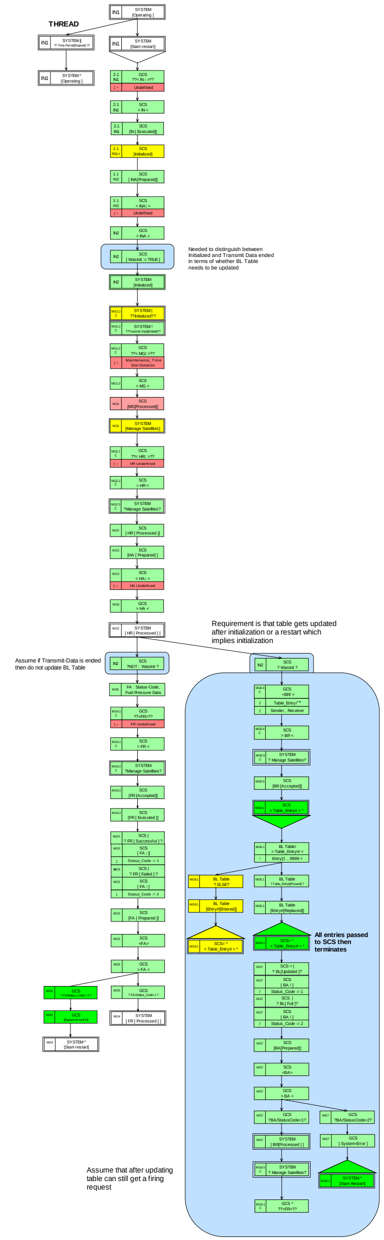
INT-03 (Integrating 2.2.1 Report Satellite Health)



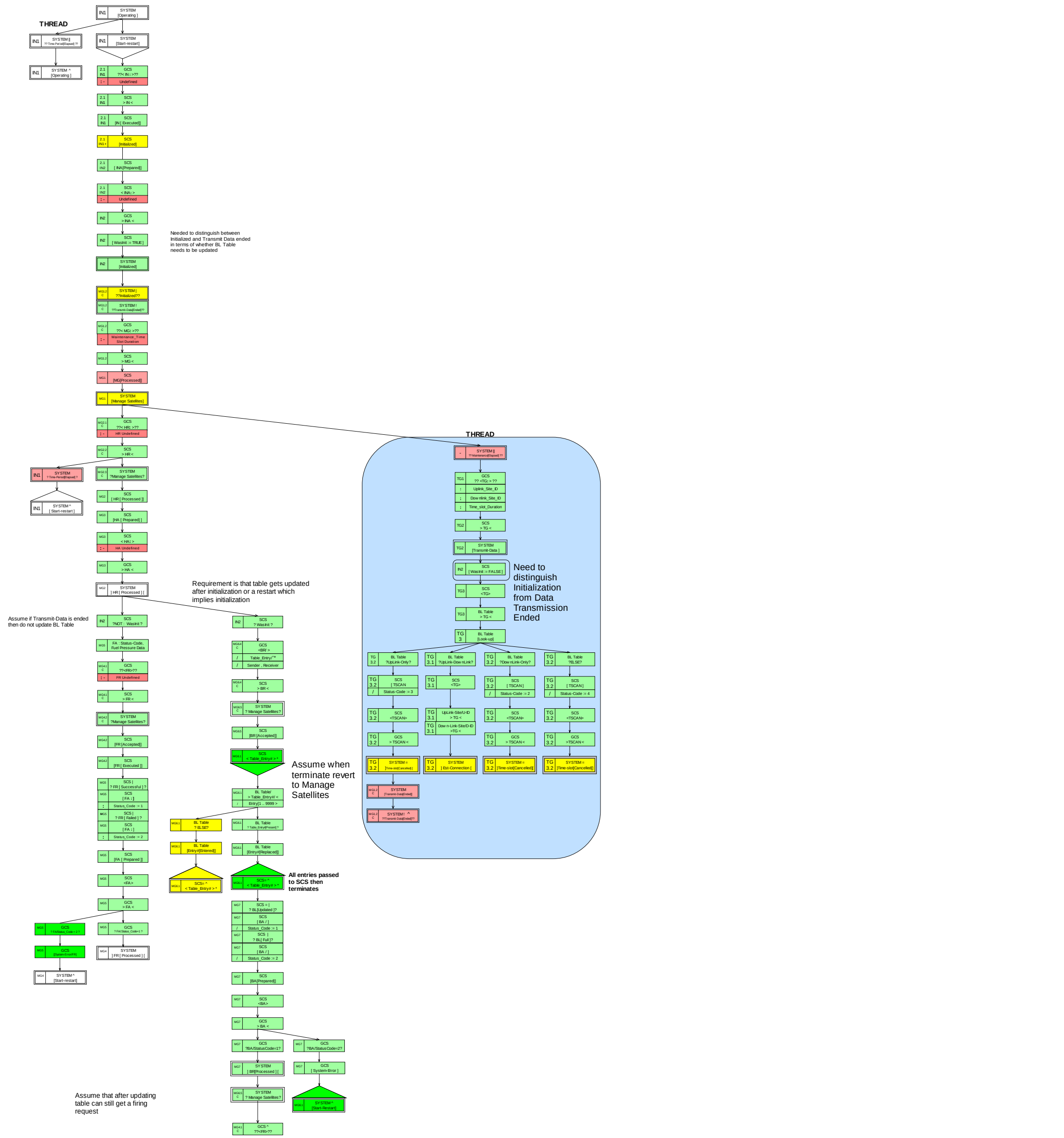
INT-04 (Integrating 2.2.2 Report Satellite Health)



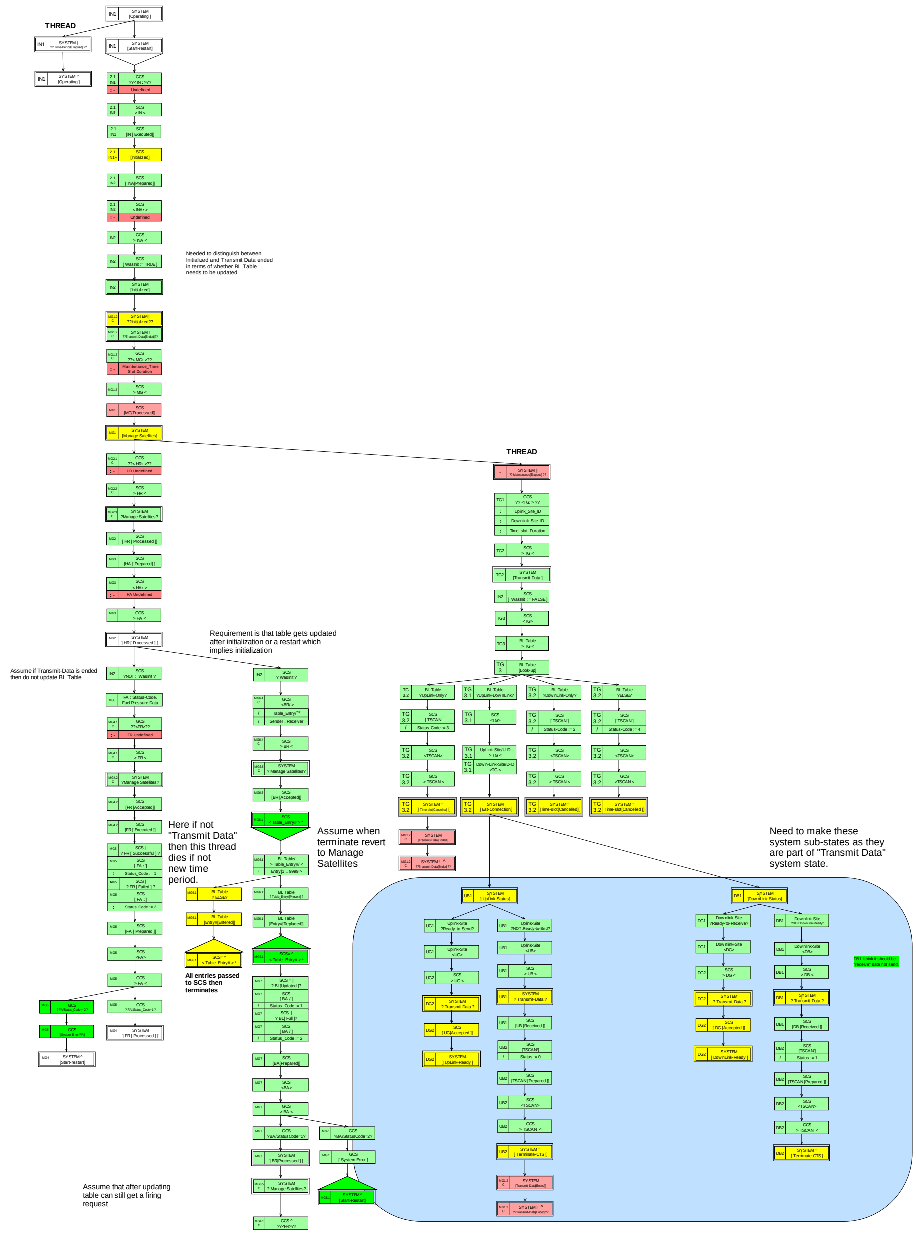
INT-05 (Integrating 2.2.3 Maintain BL Table)



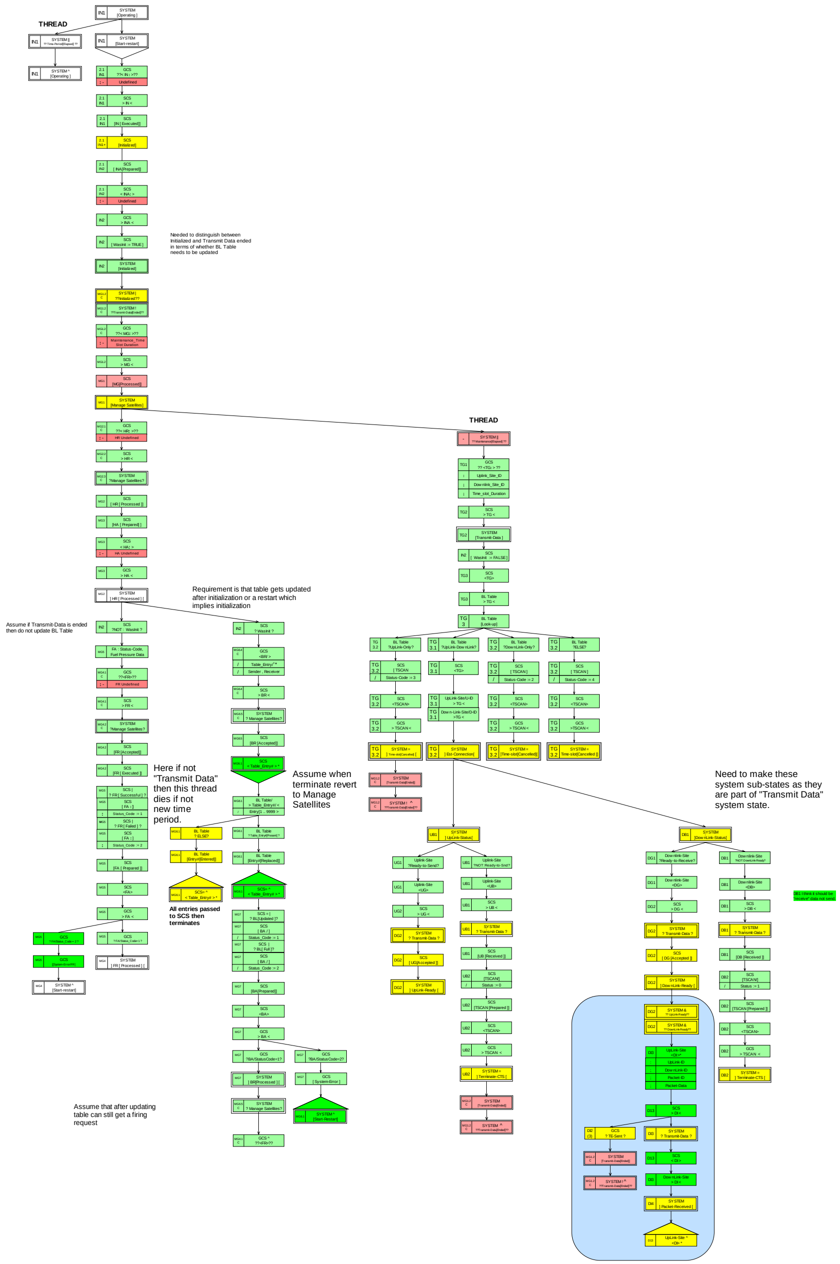
INT-06 (Integrating 2.3.1 Establish Uplink and Downlink Site Connection)



INT-07 (Integrating 2.3.2 Report Site Readiness)

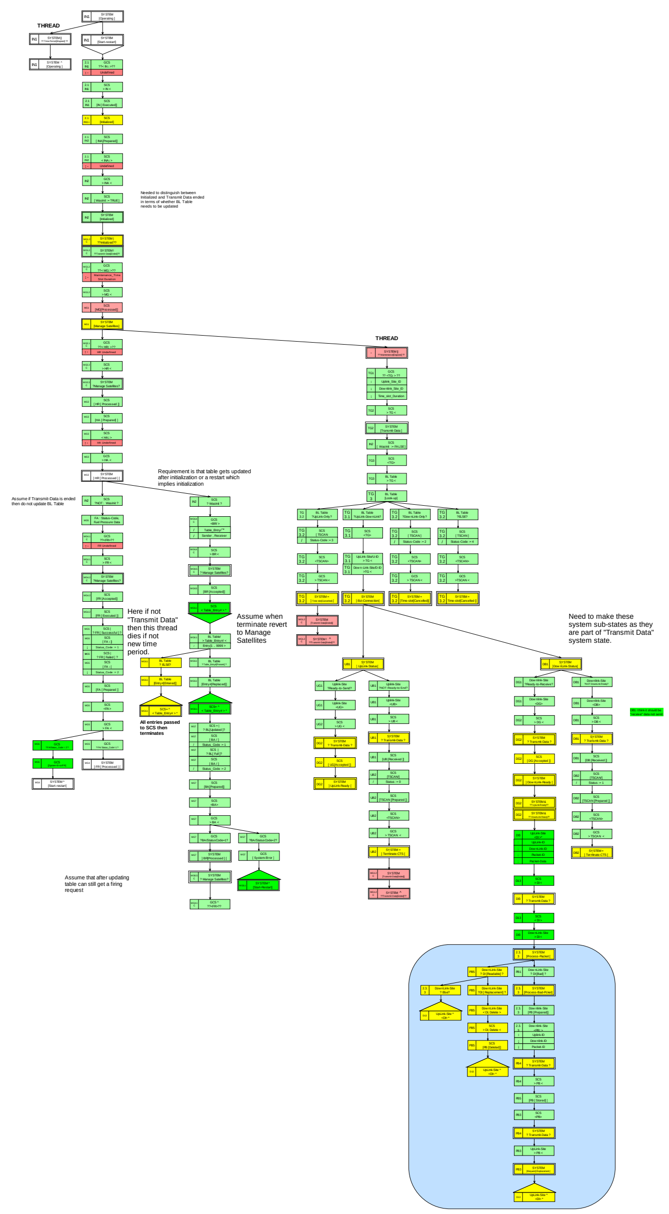


INT-08 (Integrating 2.3.2 Uploading-Downloading Packets)

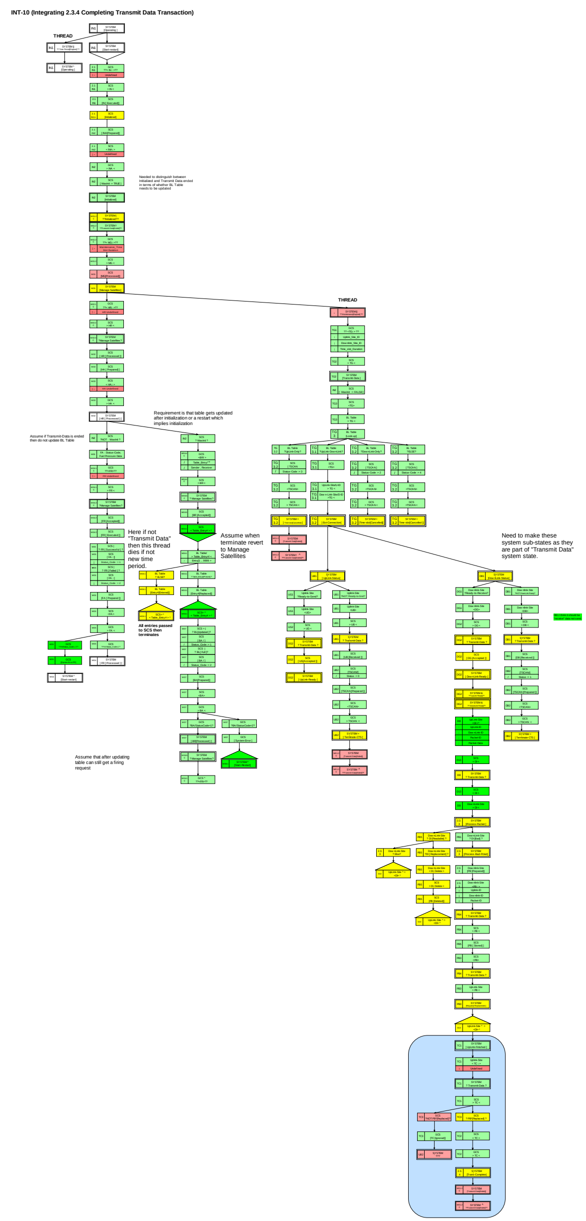


INT-09 (Integrating 2.3.3 Handling Data-packet Errors)

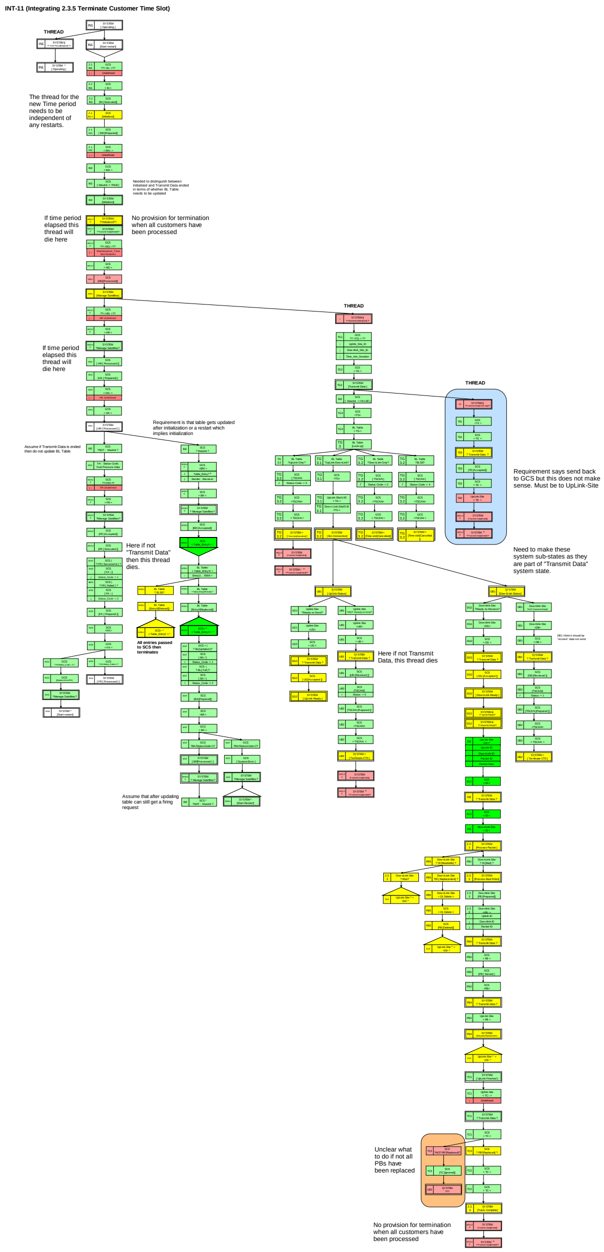
INT-09 (Integrating 2.3.3 Handling Data-packet Errors)



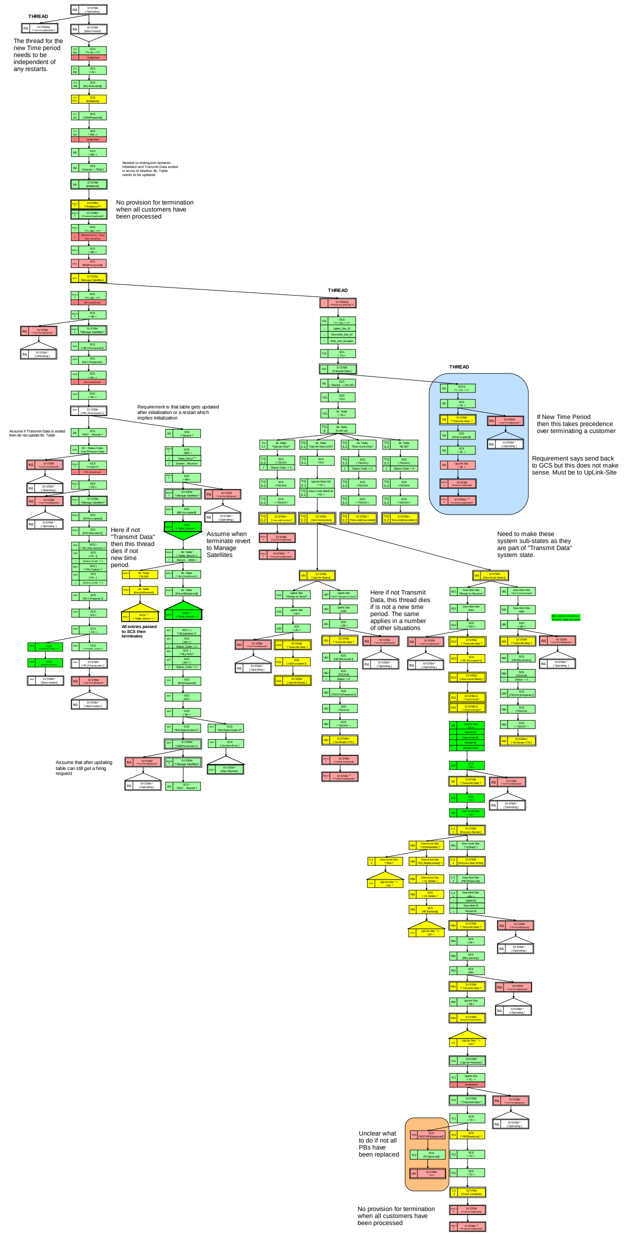
INT-10 (Integrating 2.3.4 Completing Transmit Data Transaction)



INT-11 (Integrating 2.3.5 Terminate Customer Time-slot)



INT-11 - Final Design



Integrated Views - Advantages

Integrated View => Understanding

The process of creating an integrated behavior tree does a sophisticated re-ordering of the original textual information needed so that it can be understood <u>deeply</u>, <u>accurately</u> and <u>quickly</u> without taxing and exceeding the limits of our short-term memory.

Integration => Defect Detection

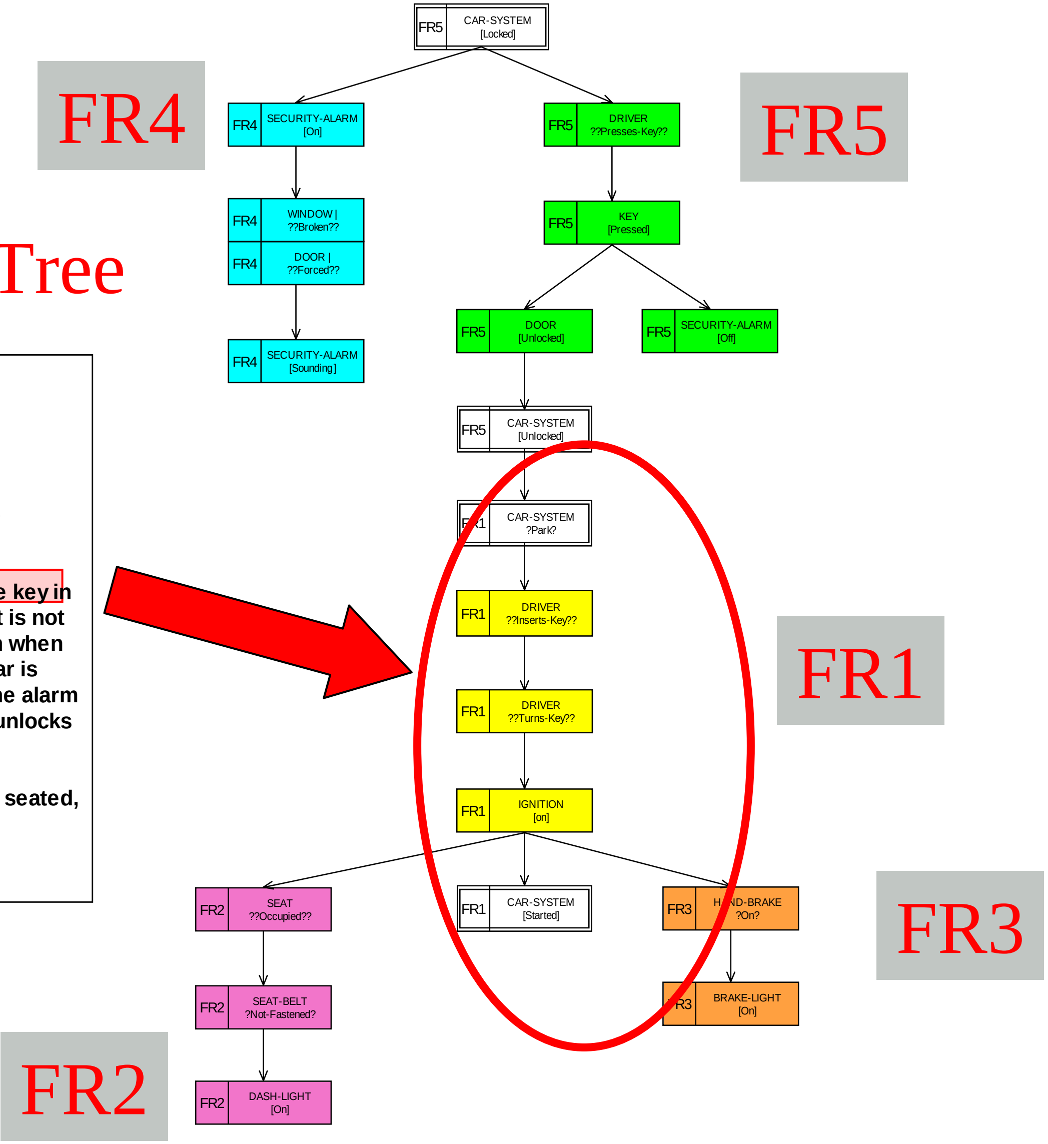
A second important advantage of the IBT representation is that the process of translation and integration uncovers incompleteness, inconsistency and redundancy <u>defects</u> in the original text that are otherwise very hard to discover because of the limitations of our short-term memory.

Integrated View => Direct Traceability to Text

NOTE:

First sentence – in text

Near bottom of Behavior Tree



PROBLEM STATEMENT

It is required to do an analysis and high-level design for part of the functional requirements for a computerised Car System with an automatic gear-shift. The following details apply:

The car can only be started if it is in the PARK state when the driver inserts the key in the ignition and turns it on. A dashboard light remains on if the driver's seatbelt is not fastened when the driver is seated and the ignition is on. If the handbrake is on when the ignition is on, the brake-light turns on. The security alarm is on when the car is locked, and if anyone tries to break in by breaking a window or forcing a door the alarm will sound. When the driver, on approaching the car, presses the key-button it unlocks the door and turns the security alarm off.

Model the whole process from the driver pressing the key-button, to the driver seated, to having started the engine.

FR1

Text ➔ Integrated Behavior Tree (IBT)

Integrated View => Direct Traceability to Text

NOTE:

First sentence – in text

Near bottom of Behavior Tree

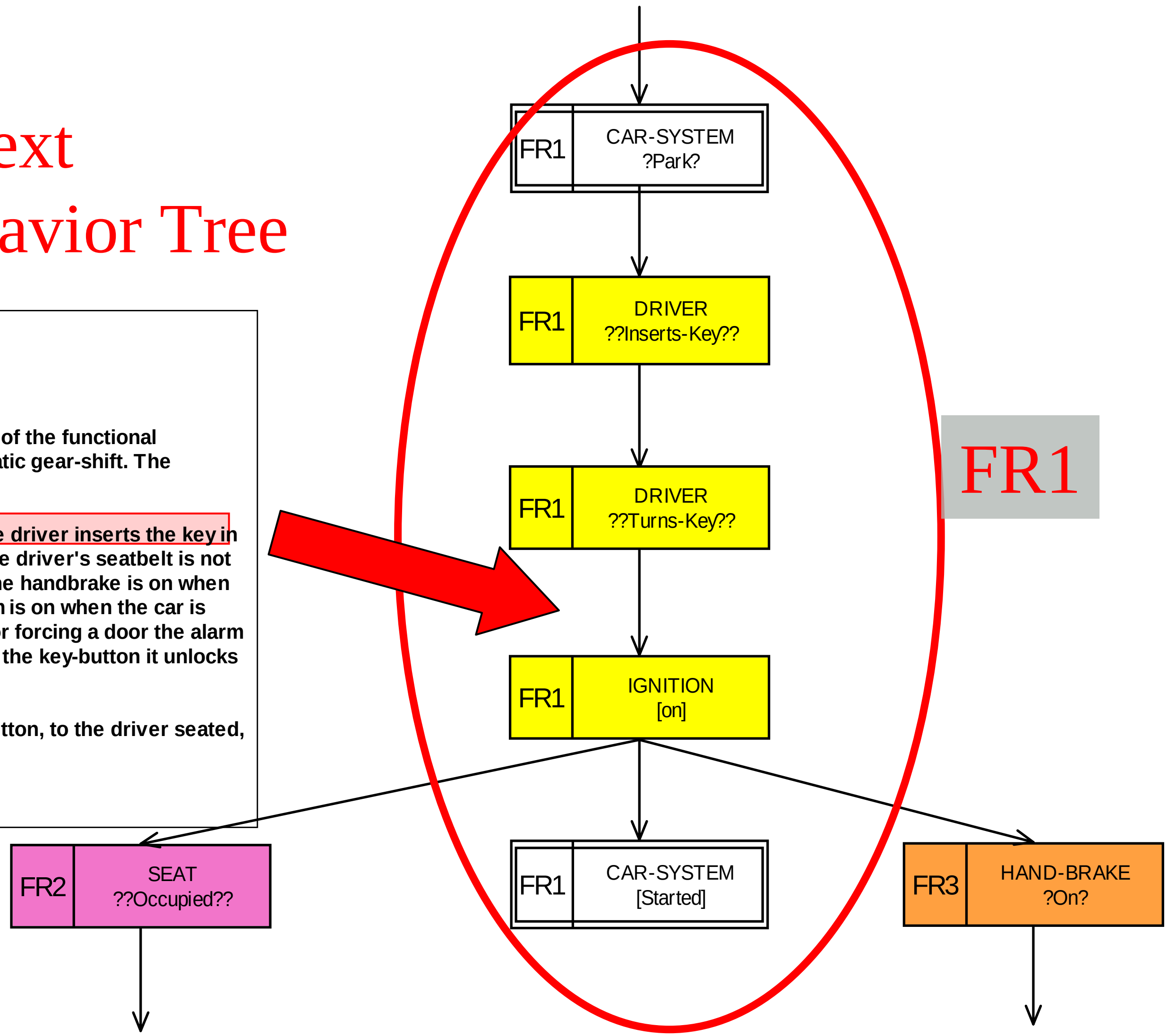
PROBLEM STATEMENT

It is required to do an analysis and high-level design for part of the functional requirements for a computerised Car System with an automatic gear-shift. The following details apply:

The car can only be started if it is in the PARK state when the driver inserts the key in the ignition and turns it on. A dashboard light remains on if the driver's seatbelt is not fastened when the driver is seated and the ignition is on. If the handbrake is on when the ignition is on, the brake-light turns on. The security alarm is on when the car is locked, and if anyone tries to break in by breaking a window or forcing a door the alarm will sound. When the driver, on approaching the car, presses the key-button it unlocks the door and turns the security alarm off.

Model the whole process from the driver pressing the key-button, to the driver seated, to having started the engine.

FR1

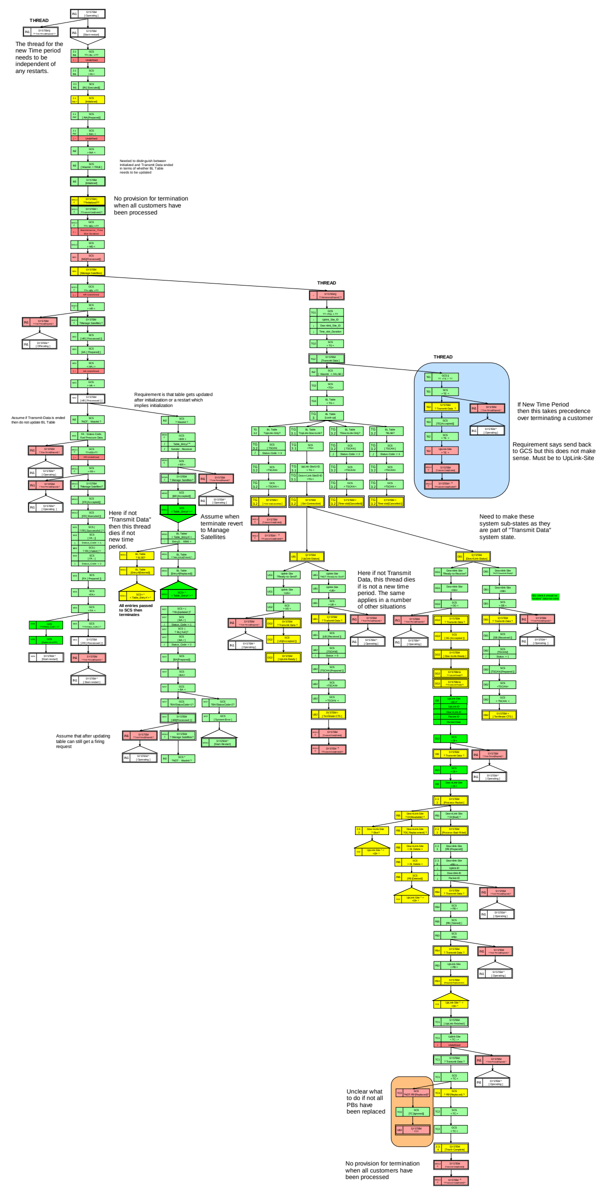


Text ➔ Integrated Behavior Tree (IBT)

Text ➔ Integrated-Behavior-Tree (IBT)

Satellite Control System

Integrated View



Compared with

23-page document of functional requirements

Scaleability => Translation + Integration

- We have applied the processes of requirements translation and integration to a large number of systems – the largest system we have worked on had 1500 requirements.

- We have been encouraged by the results we have obtained from these trials.

- It is clear that we need a tool-set that makes it practical for teams of people to apply the method.

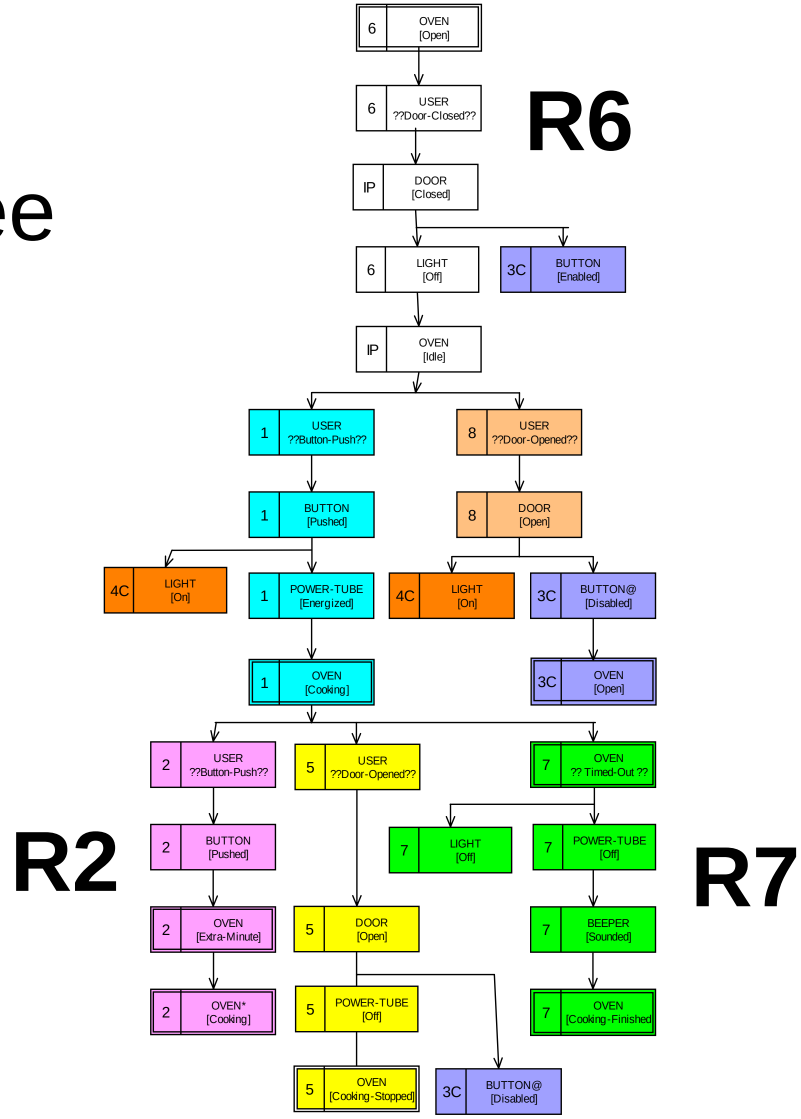
Integrated Views

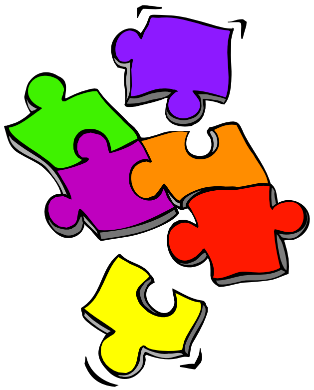
- Emergent Properties

Requirements View

Integrated Behavior Tree

Result of integrating eight functional requirements





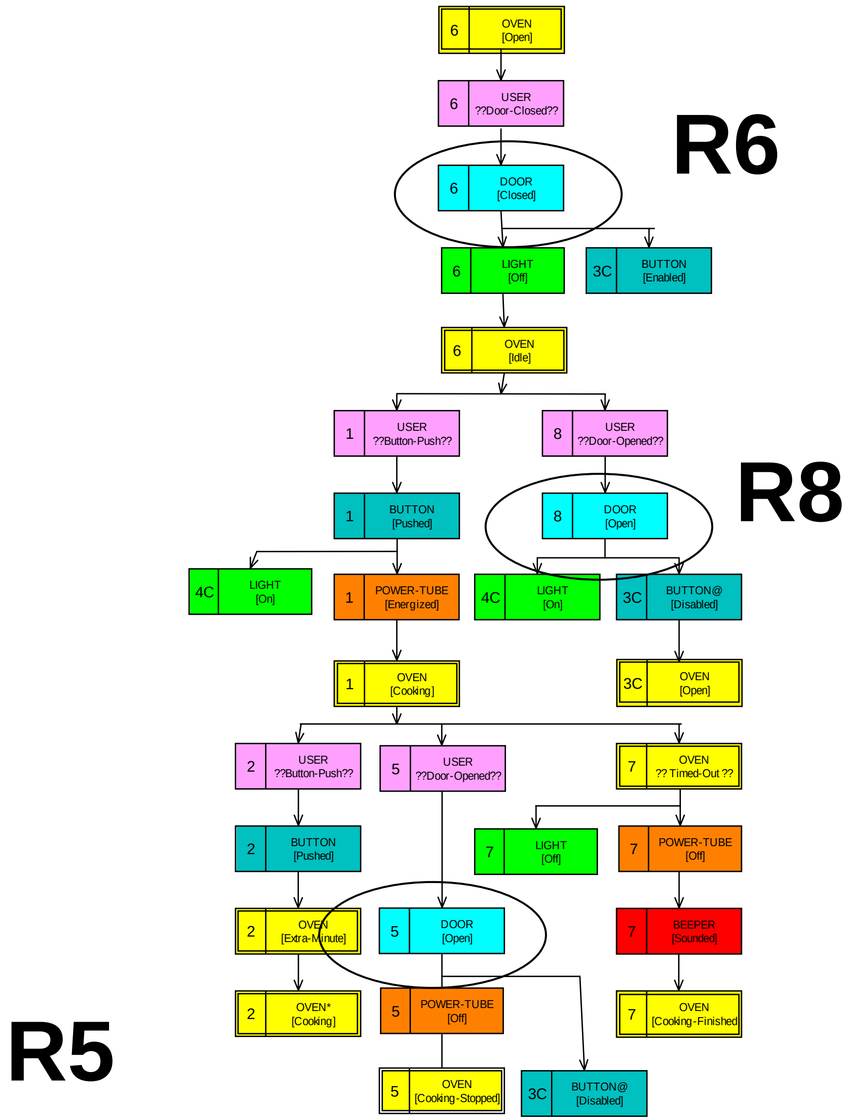
Functional Requirements are **Localized** in IBT

Component View

Integrated Behavior Tree

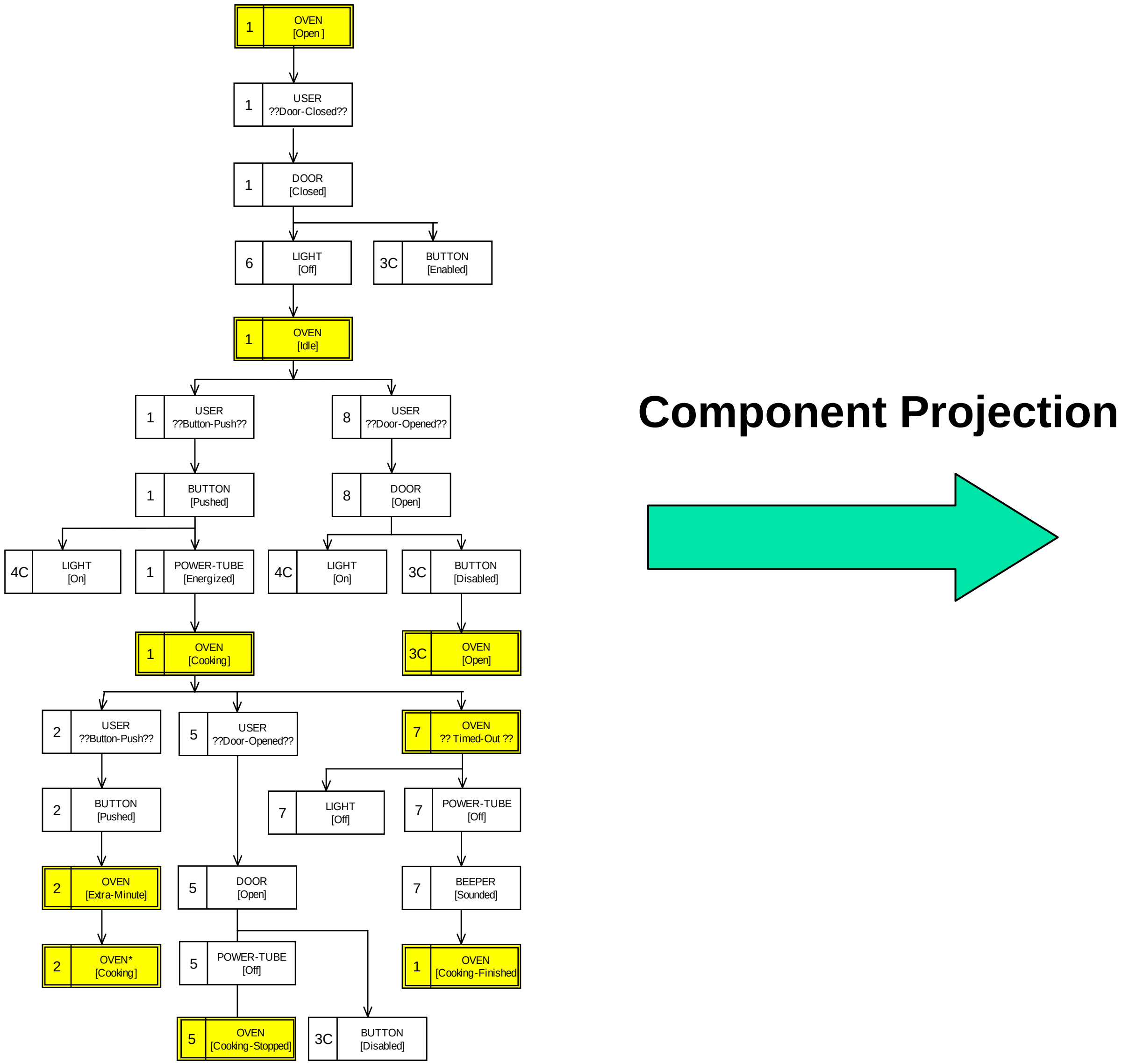
Result of integrating eight functional requirements

= **DOOR**



Components are **Dispersed** across requirements

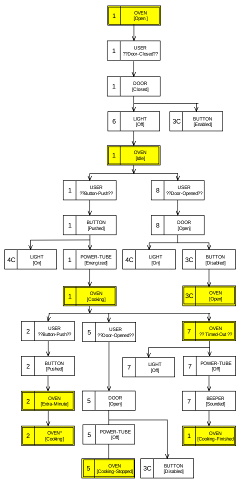
Component Behavior Projection



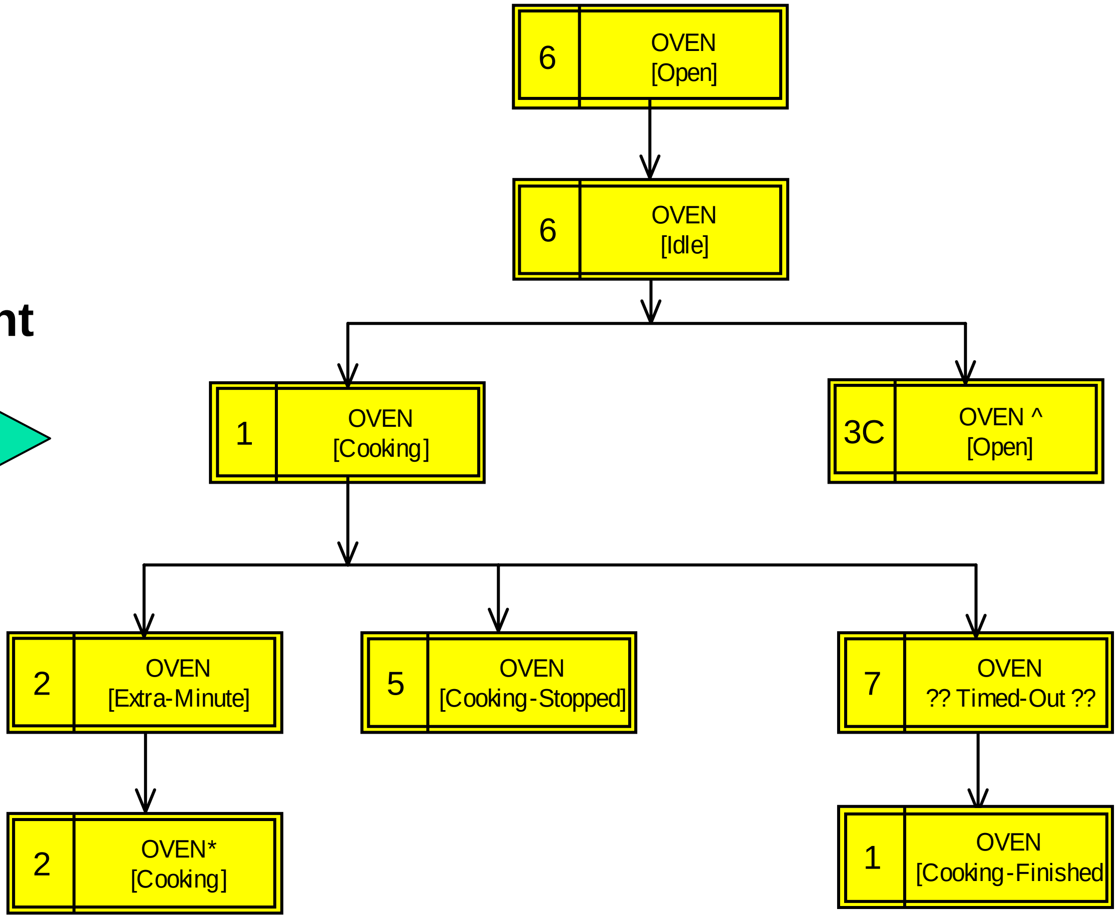
Oven Component = System Component

Component Behavior Projection

System Component Projection

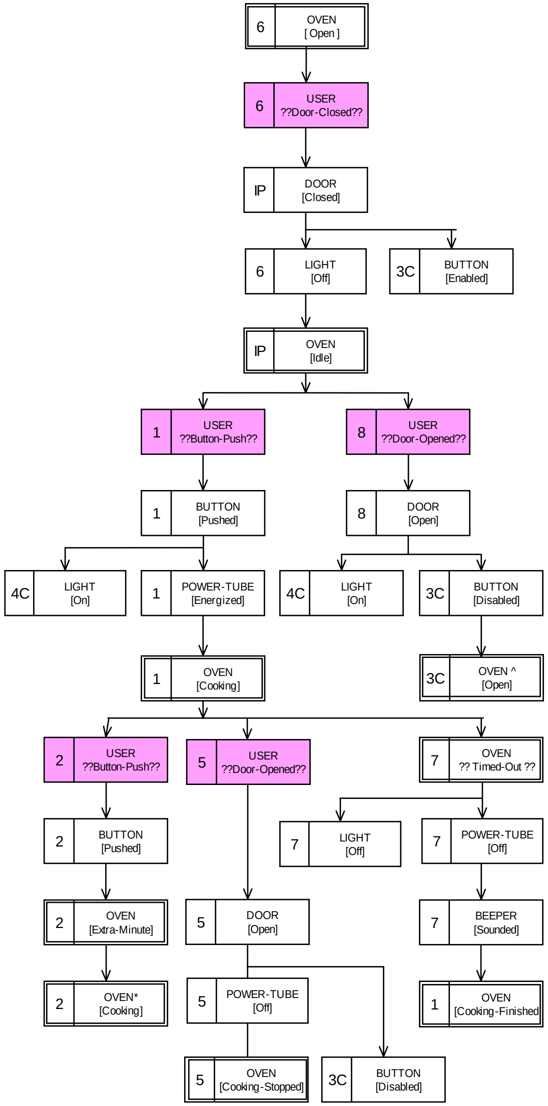


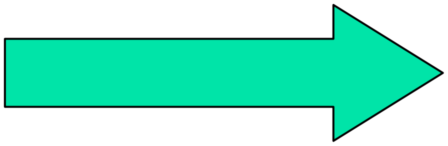
Component Projection



Oven Component = System Component

Component Behavior Projection

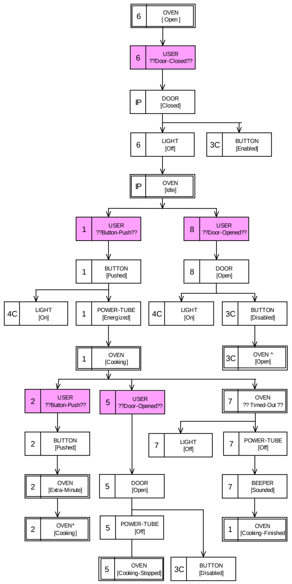




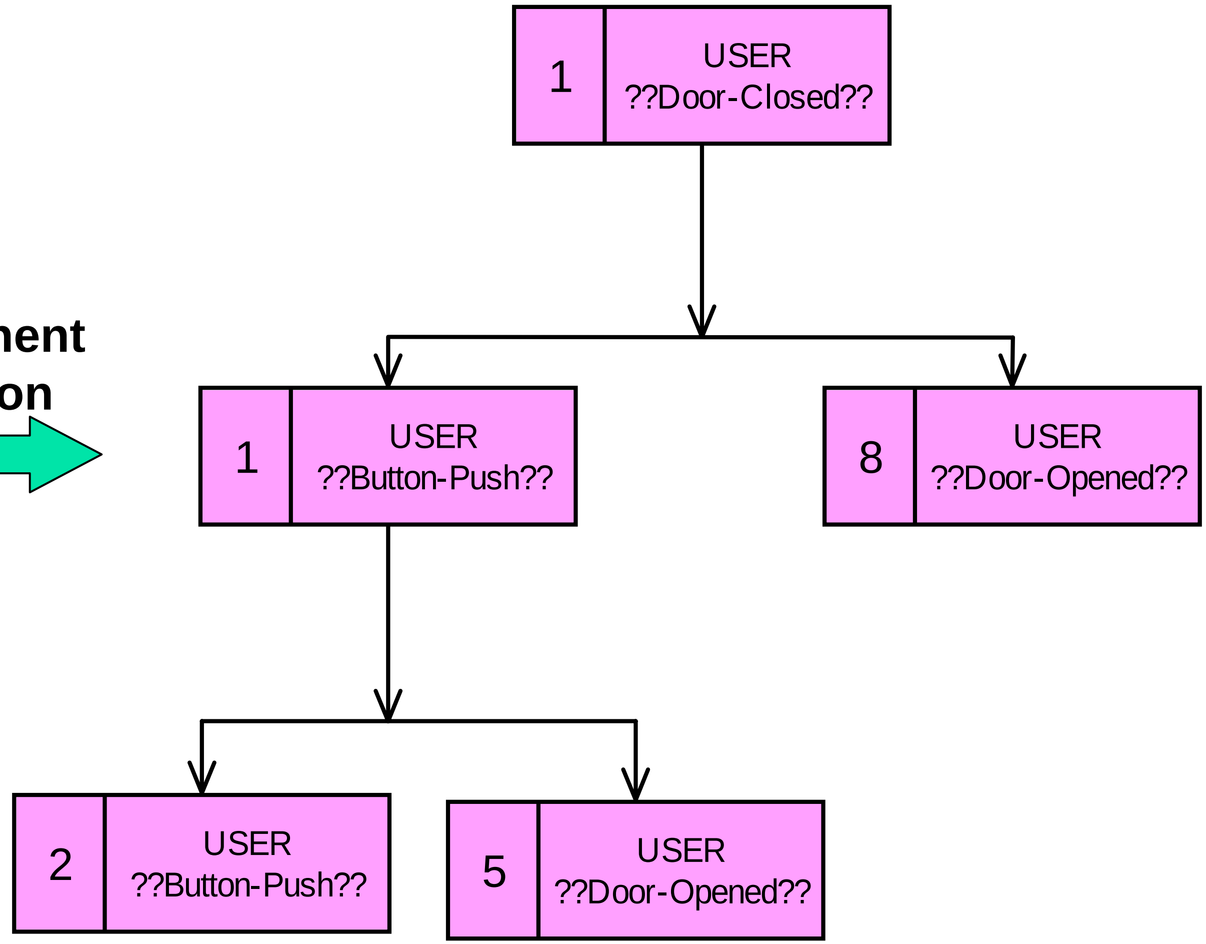
User-Interface Component

Component Behavior Projection

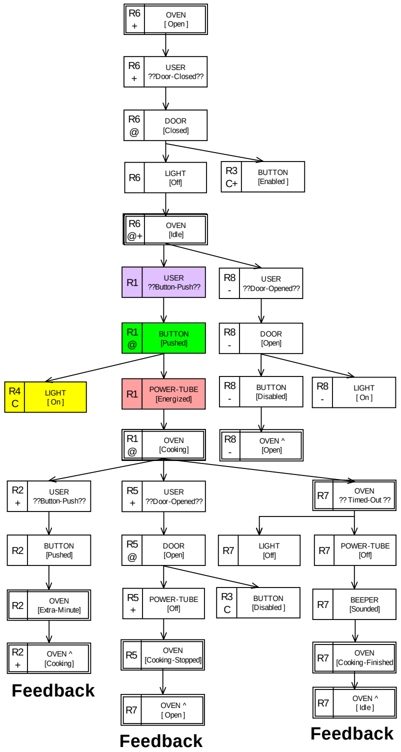
USER Component Projection



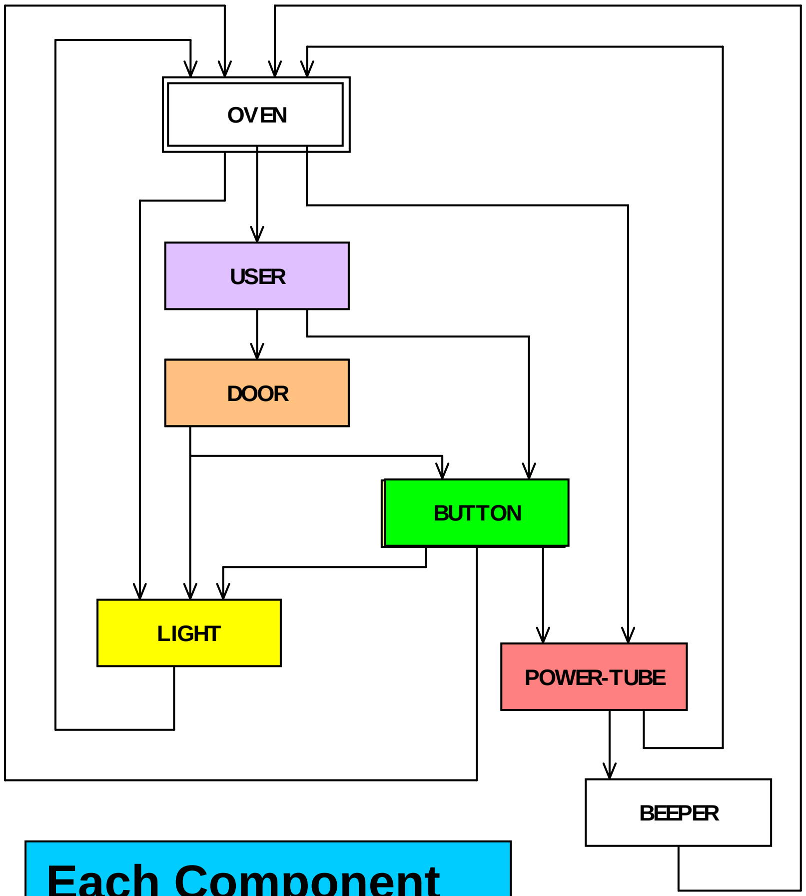
Component Projection



Architecture Transformation



All behaviors merged in together



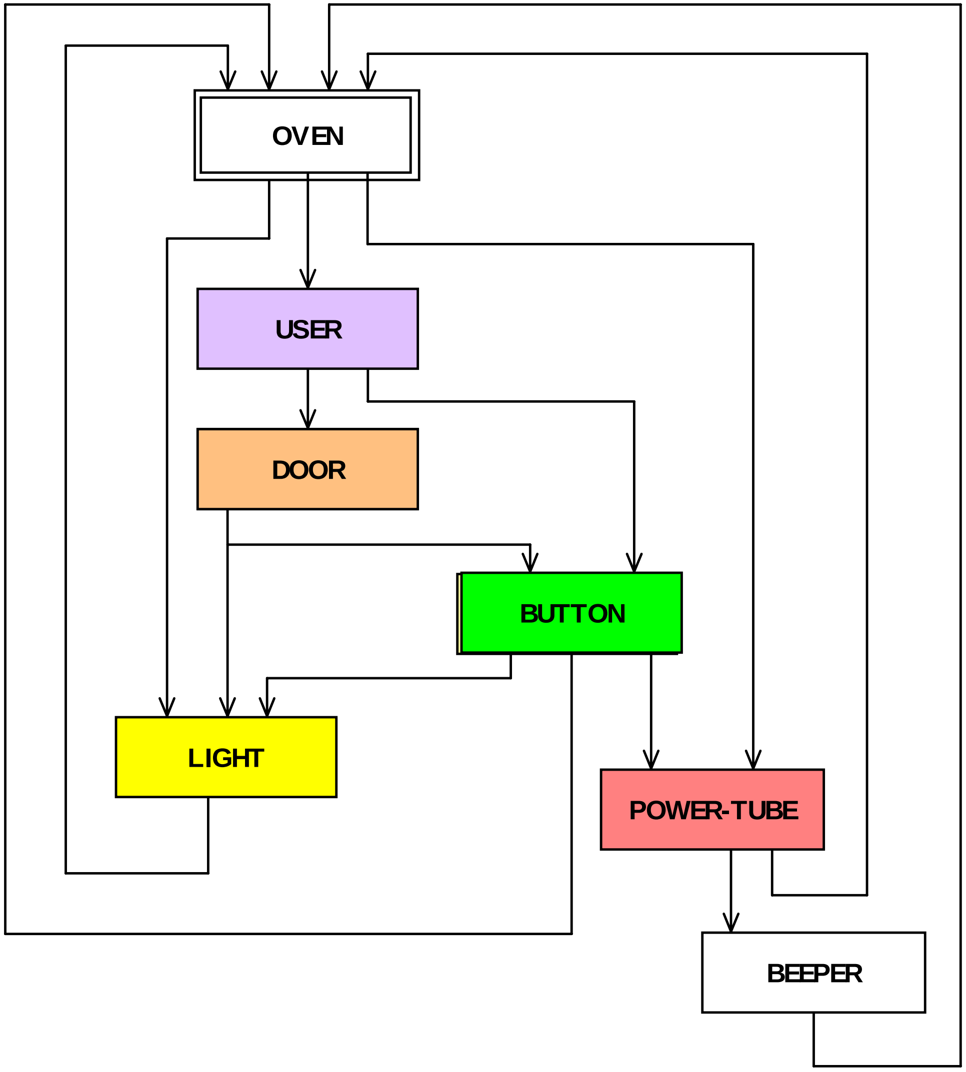
Each Component Occurs <u>MANY</u> times

Each Component Occurs just <u>ONCE</u>

System – Network of Component Interactions

Architecture Transformation

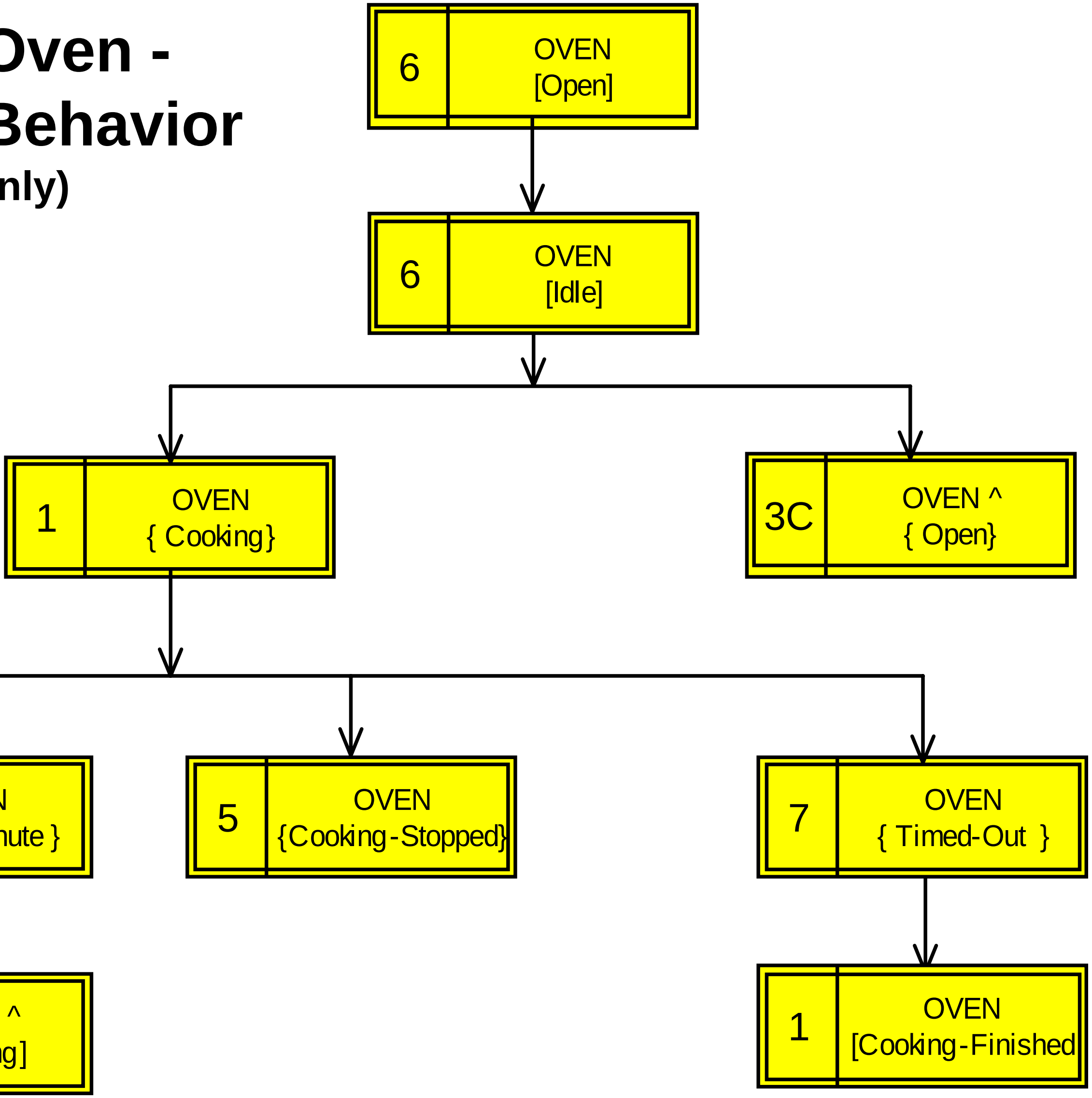
All behaviors merged in together



System – Network of Component Interactions

High-Level Behavior

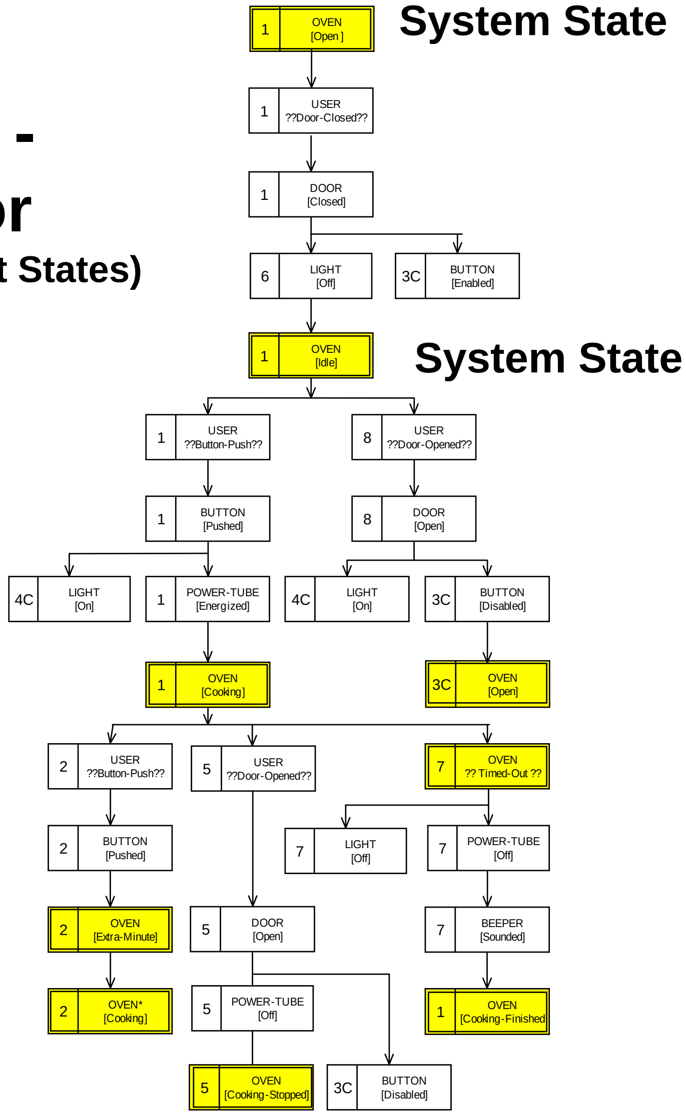
Microwave Oven -
High-Level Behavior
(System States Only)



Detailed Behavior

Microwave Oven - Detailed Behavior

(System States + Component States)

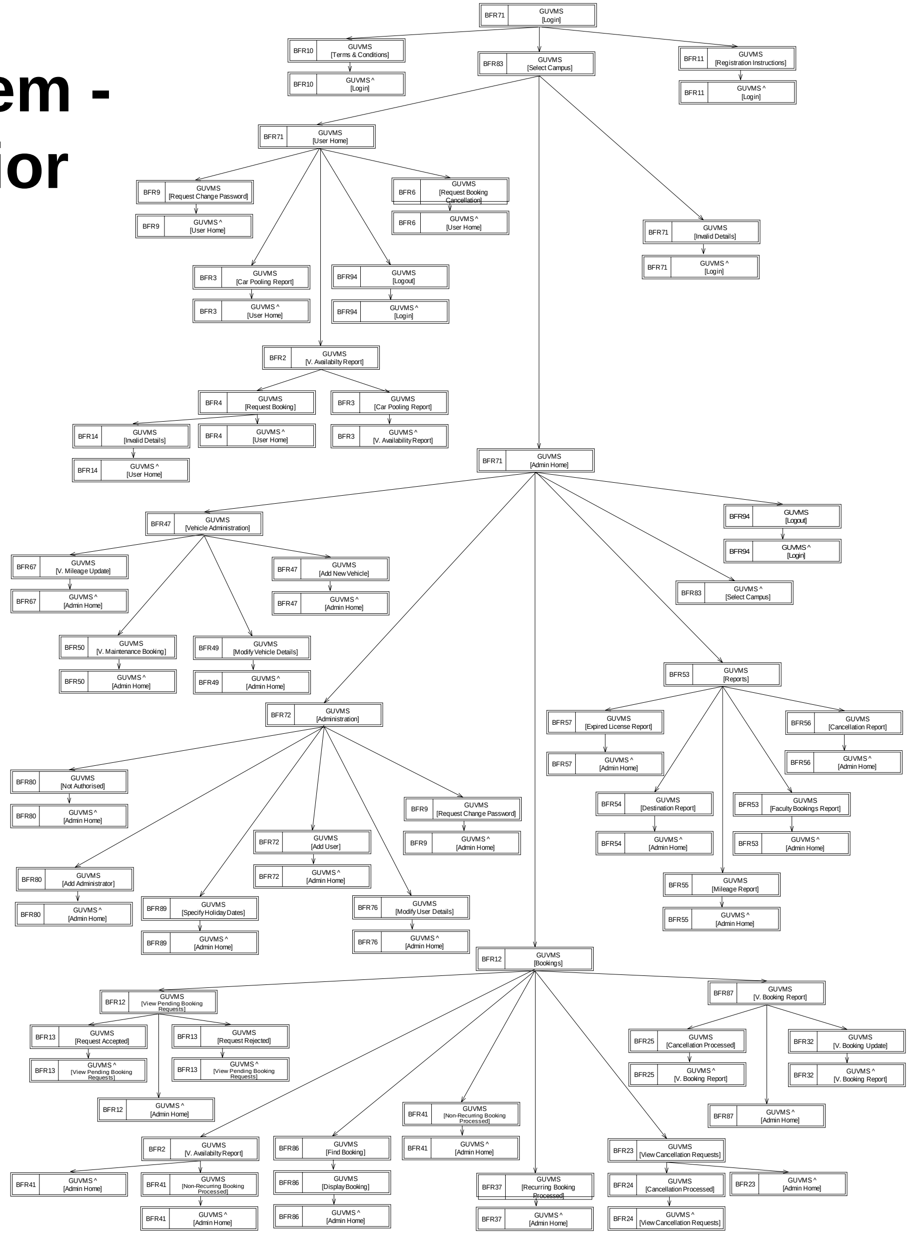


System-states are embedded

High-Level Behavior

Car-booking System - High-Level Behavior

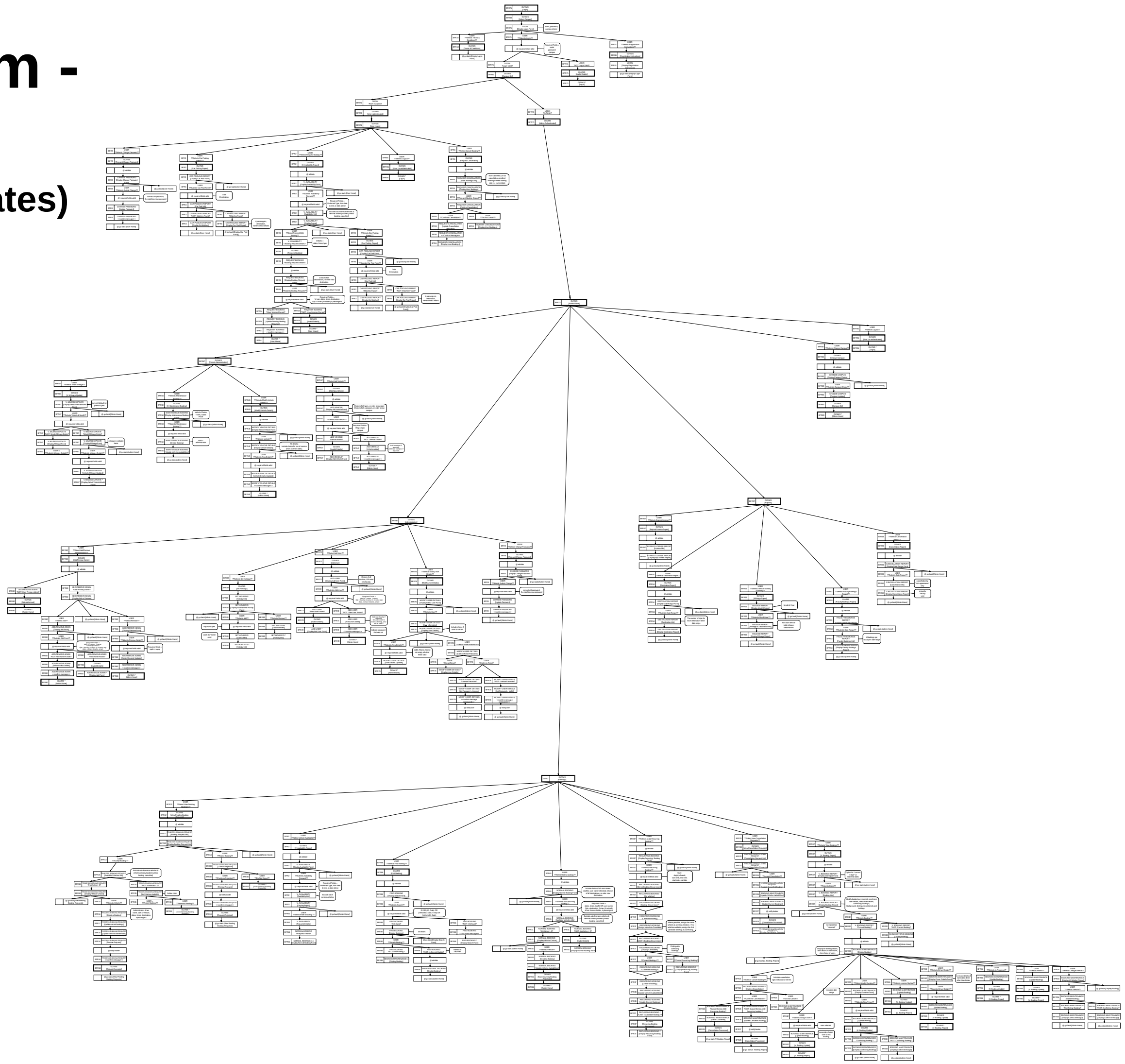
(System States Only)



Detailed Behavior

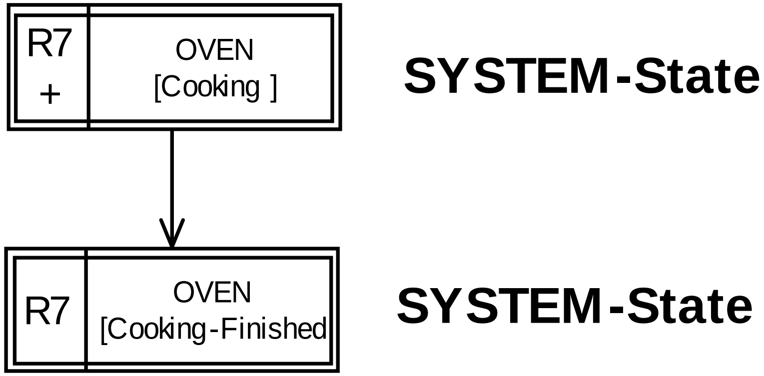
Car-booking System - Detailed Behavior

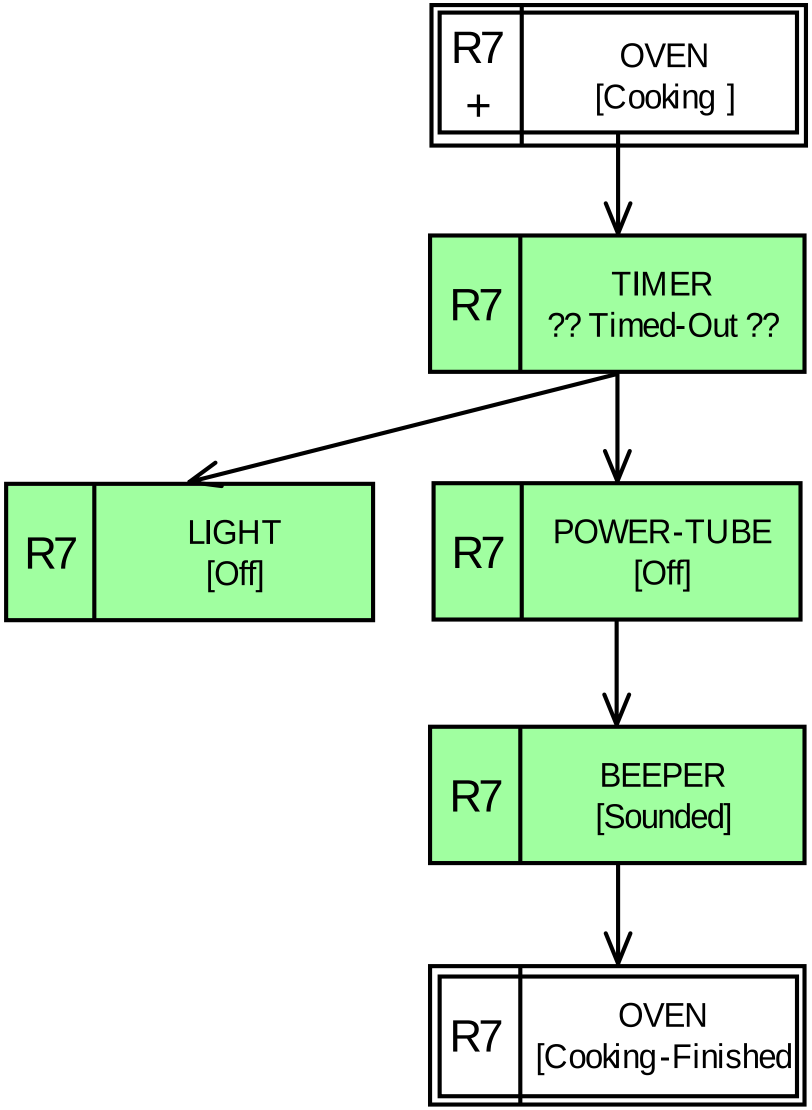
(System States + Component States)



System-states are embedded

Behavior – Including System States





Abstract versus expanded description of behavior

Actions on Integrated Views

Integrated View => Specification

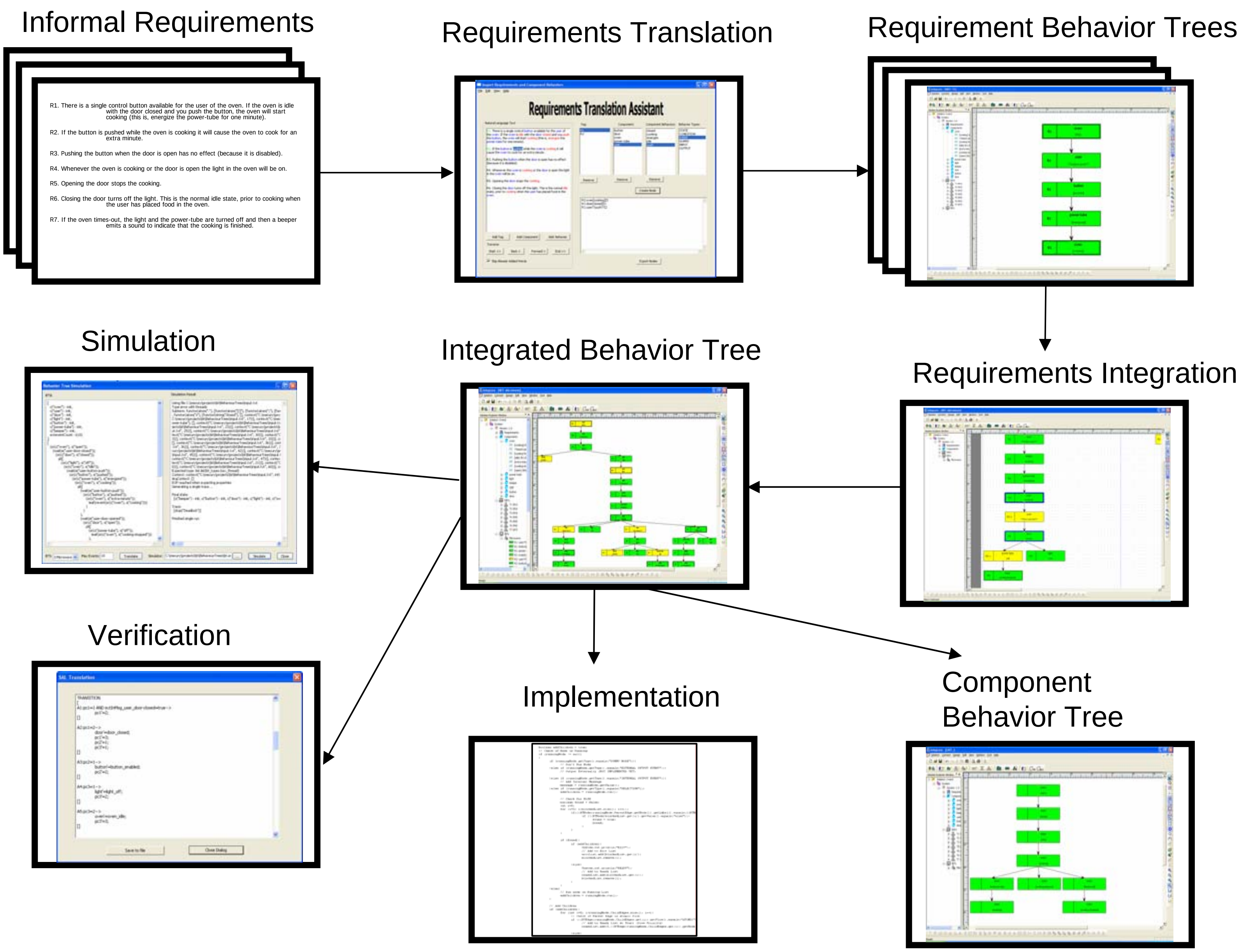
Actions On

- Inspection/defect detection
- Model-checking
- Defect correction
- Refinement => Specification => Design

And then:

- Component Behavior Projections
- Component Interface Specifications
- Architecture Transformation
- Integration specification
- Code Generation

Building Dependable Systems



END

Part 1

Where Are We Up To

Scale & Complexity ✓

Build Right System ✓

Imperfect Knowledge ×

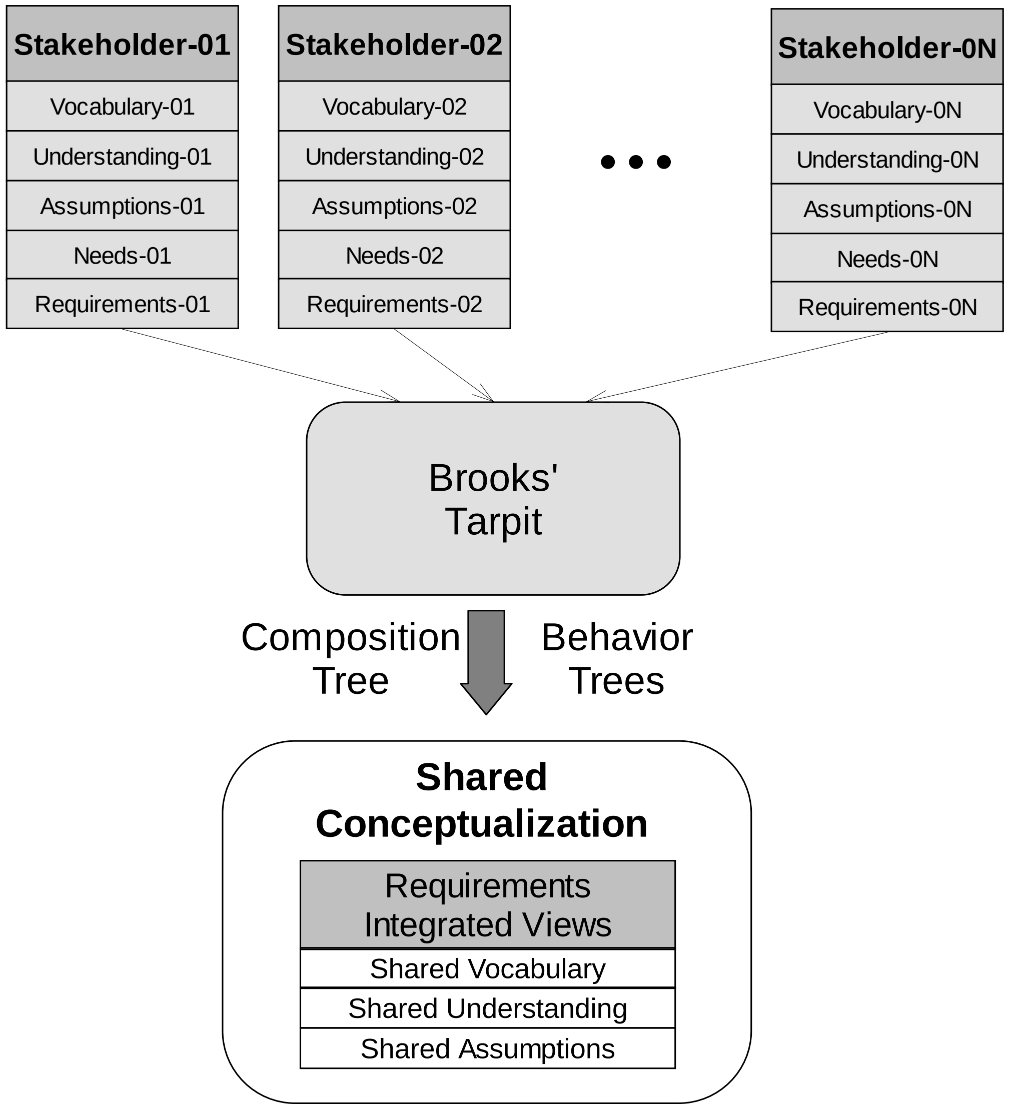
Productivity ×

Integrated View - Integrated Behavior Tree ✓

Tackling

Imperfect Knowledge

Problem 2 – Imperfect Knowledge

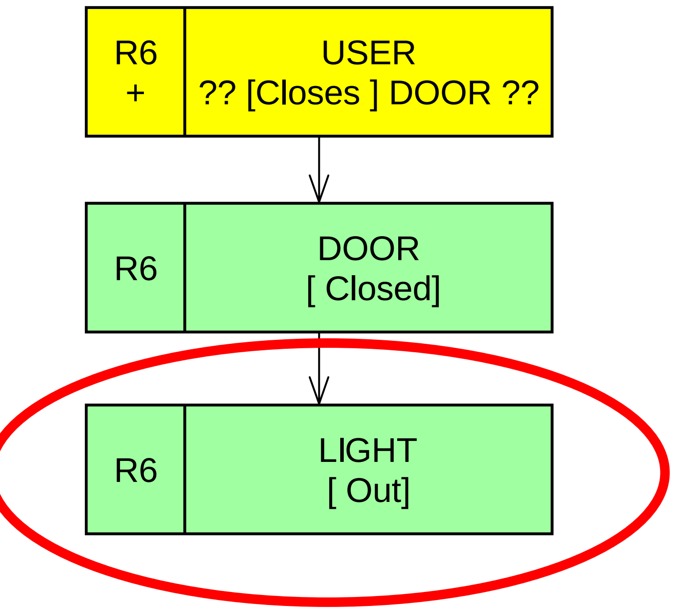


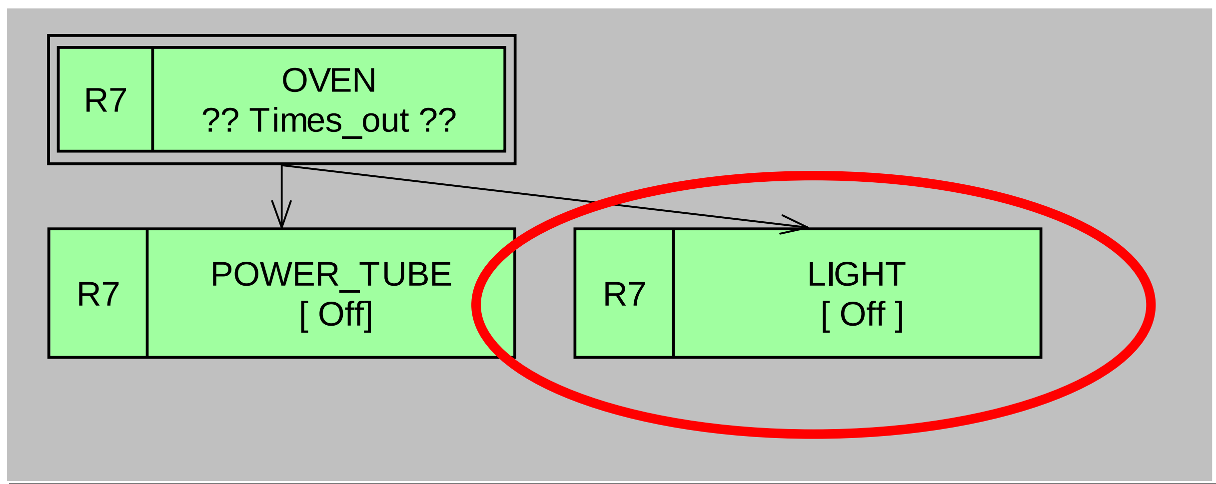
Inconsistencies among stakeholders

Imperfect Knowledge - Aliases

R6. If you close the door, the light goes out. This is the normal configuration when someone has just placed food inside the oven but has not yet pushed the control button.

R7. If the oven times out, it turns off both the power tube and the light. It then emits a warning beep to tell you the food is ready.





Independent Translation

Inconsistency

Imperfect Knowledge - Aliases

R6. If you close the door, the light goes out. This is the normal configuration when someone has just placed food inside the oven but has not yet pushed the control button.

R7. If the oven times out, it turns off both the power tube and the light. It then emits a warning beep to tell you the food is ready.

Implication

We could formalize each requirement <u>independently</u> but we would end up with an inconsistent vocabulary.

- **We have to overcome this problem.**
- **Challenge when 100s requirements**

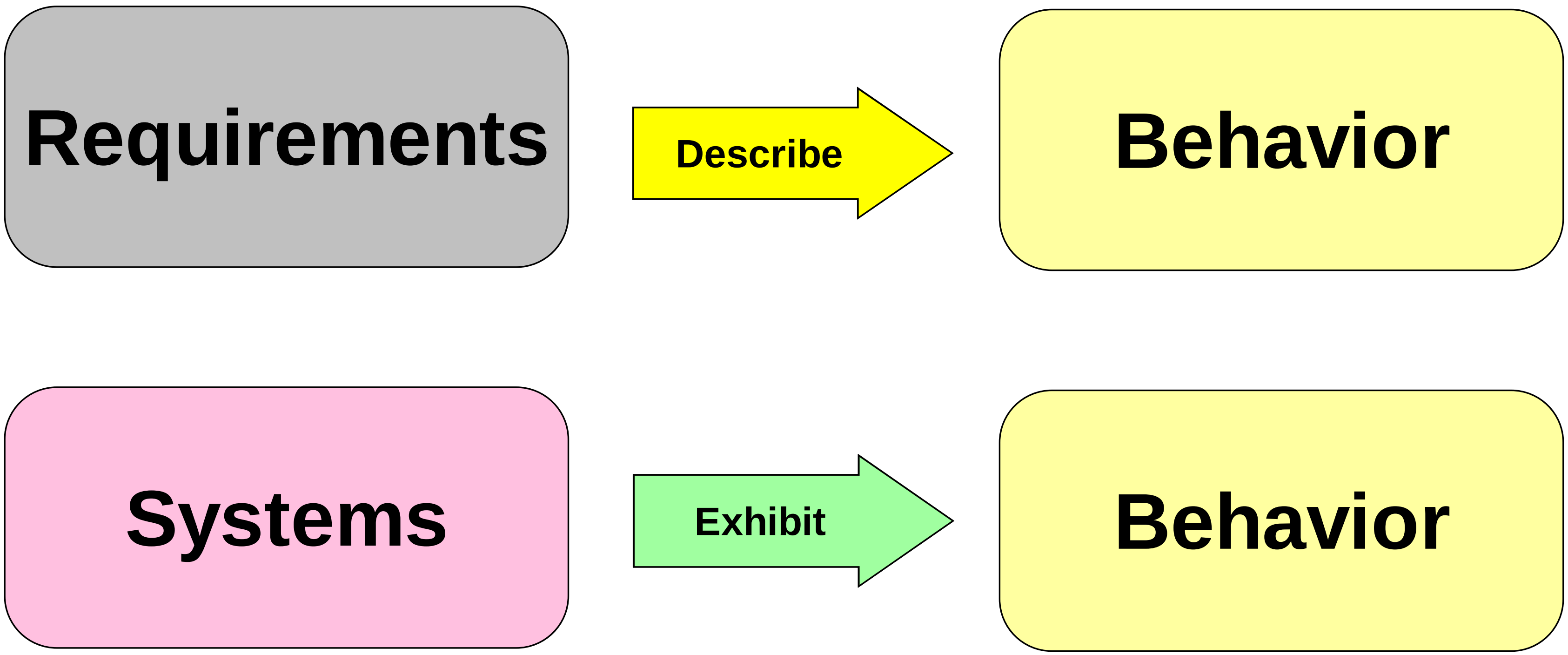
Imperfect Knowledge - Aliases

We use a second Integrated View to solve problem

Integrated Composition Tree

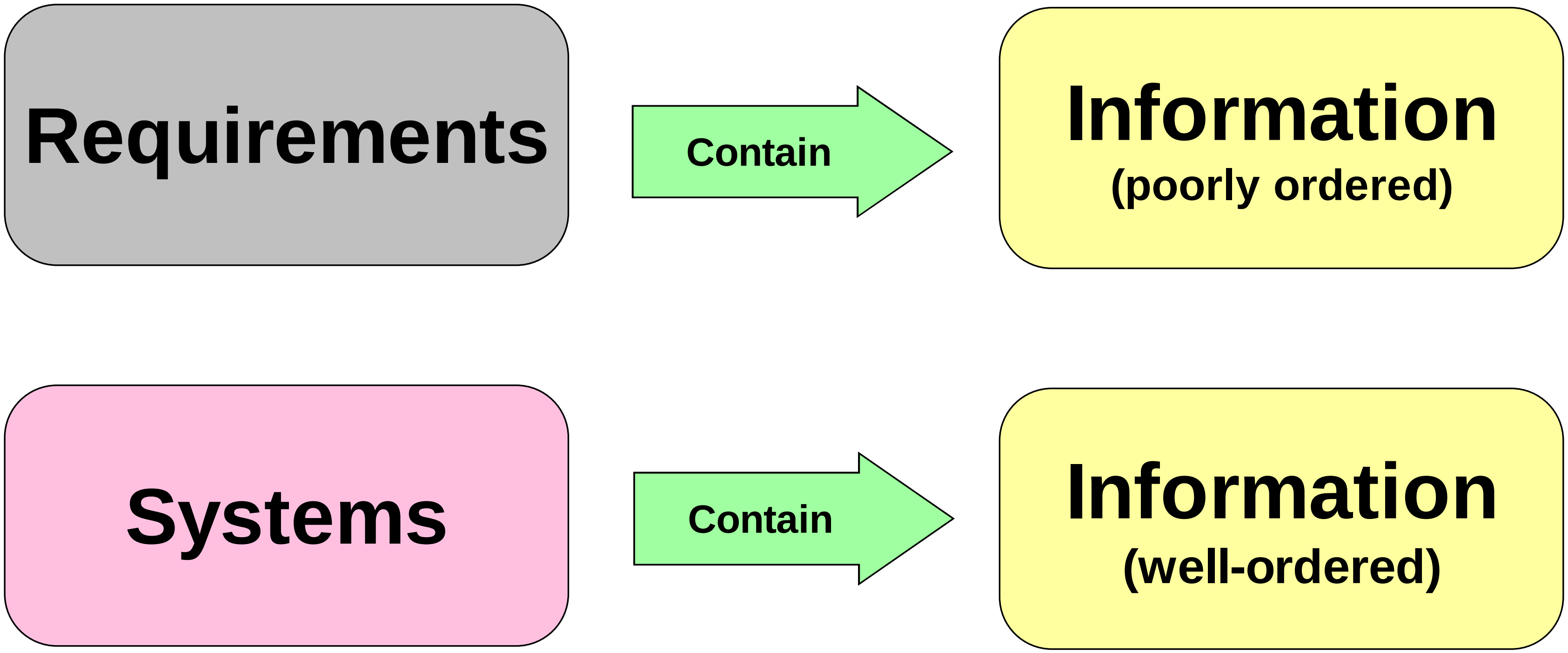
We saw earlier

Requirements ⇔ Systems ⇔ Behavior



The Link – Build systems <u>out of</u> requirements

Requirements Versus Systems



Increase Order => Remove Imperfections

Requirements Versus Systems

- **Requirements for systems contain INFORMATION.**

- **Systems that satisfy requirements contain INFORMATION**

The Link – Build systems out of their requirements

Where to Start => Understanding

- Confronted with a statement of requirements our job is to systematically and effectively increase our **understanding** of the problem to be solved.

- To increase understanding we need to create useful, usable, new order in a **repeatable**, constructive way.

- It turns out that constructing the **system composition** is probably the most effective way to do this and thereby initiate the analysis/design phase.

Composition

Composition is a concept that is widely used in a number of disciplines to provide useful summary information about an entity.

Composition

Examples

- **HOUSE: 4 bedrooms, 2 bathrooms, ...**

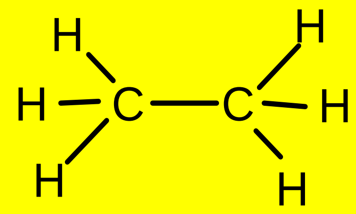
- **ETHANE: C_2H_6**

- **DICTIONARY:**

  Table: " A piece of furniture consisting of a flat top set horizontally on legs"

Relevant for analyzing/design of large systems

A Way to Look at Things - Chemistry

	Representations		Ethane Molecule
1	**COMPOSITION**	→	C_2H_6
2	**STRUCTURE**	→	
3	**BEHAVIOR**	→	Detailed description of behavior

What is Important About Composition

Composition is a fundamental property of a system.

Composition is a fundamental property of a set of functional requirements of a system.

Properties can be identified repeatably

What is Important About Composition

It should be UNIQUE for a given statement of requirements.

System Composition

The **System Composition** plays a role in system design of comparable importance to the role <u>laying the foundations</u> plays in constructing a house – it comes first and it supports all subsequent activities.

Complete Vocabulary ➔ Well-defined Property

System Composition => Built on System Vocabulary

What Composition Addresses

The problem we face in attempting to build an understanding of the components in a system is that in statements of requirements for a system, information about an individual component in the system is usually **widely spread** throughout the set of requirements.

Component composition => System Composition

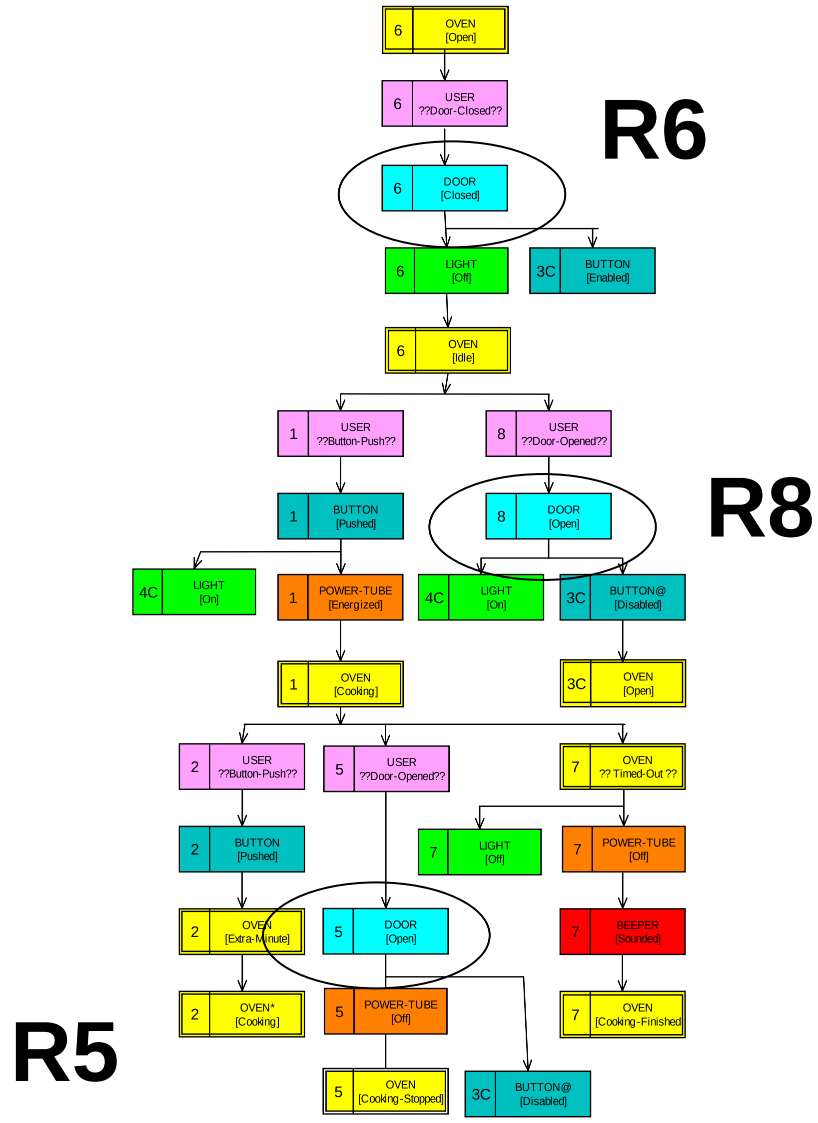
The Problem We Face

Integrated
Behavior Tree

Result of integrating eight functional requirements

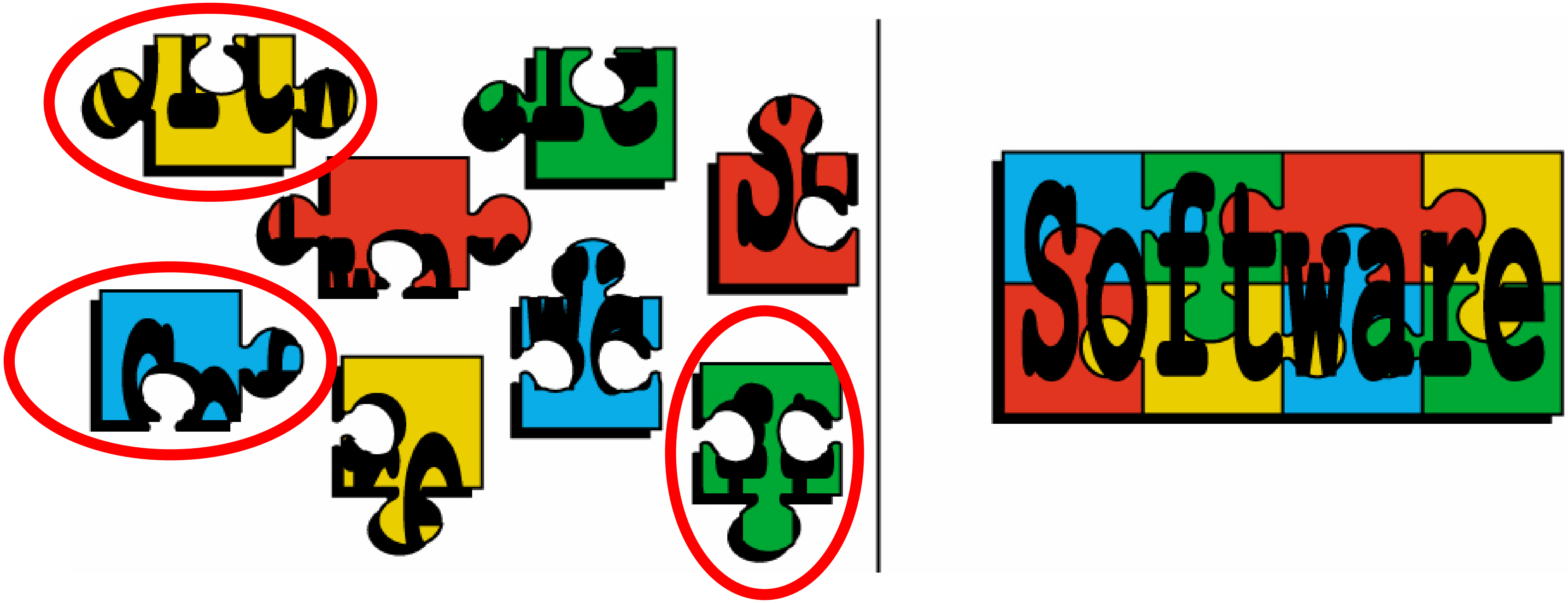
= **DOOR**

DOOR Component Mentioned in R6, R8, R5



Individual Component Info dispersed across requirements

Creating an Integrated View

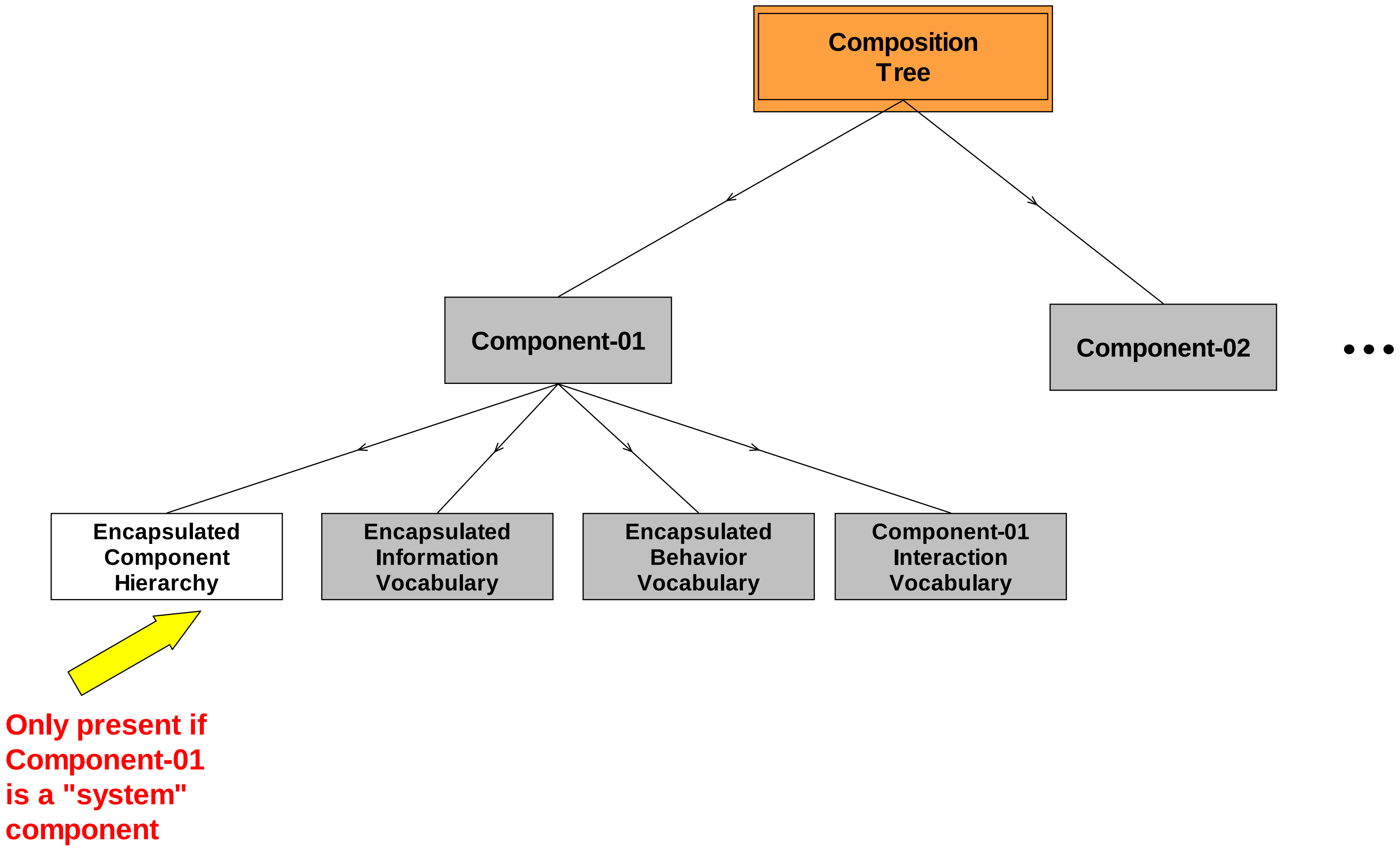


Information about “f” is spread across THREE pieces

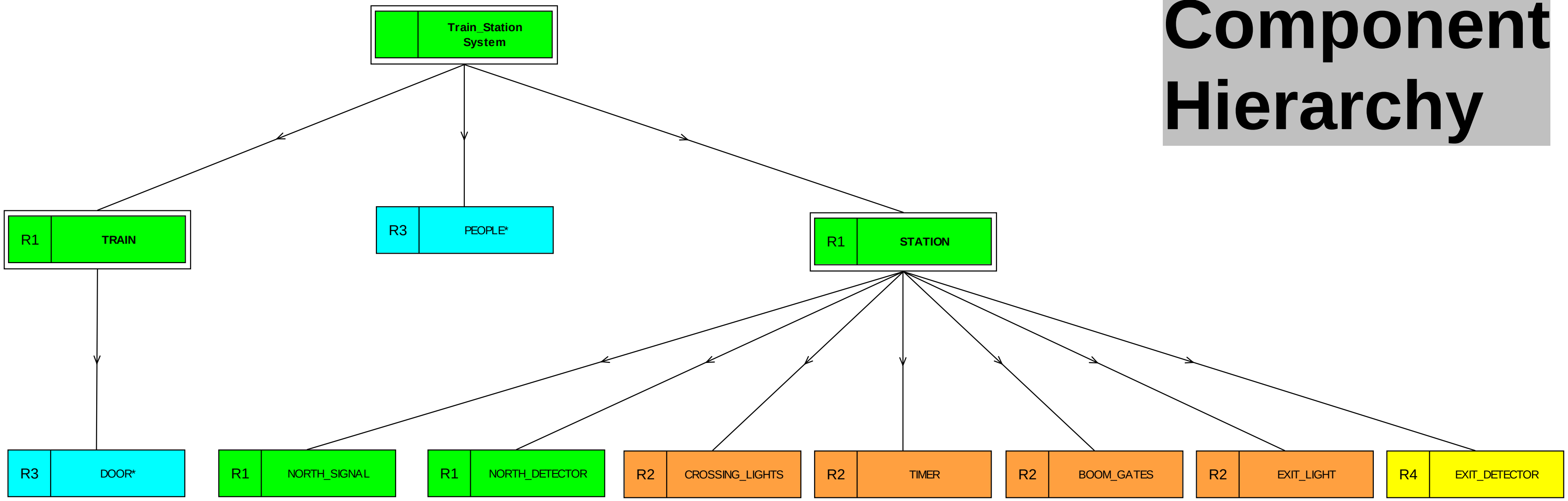
Integrated View – Component “Picture” Emerges

Composition Trees

Composition Tree Form



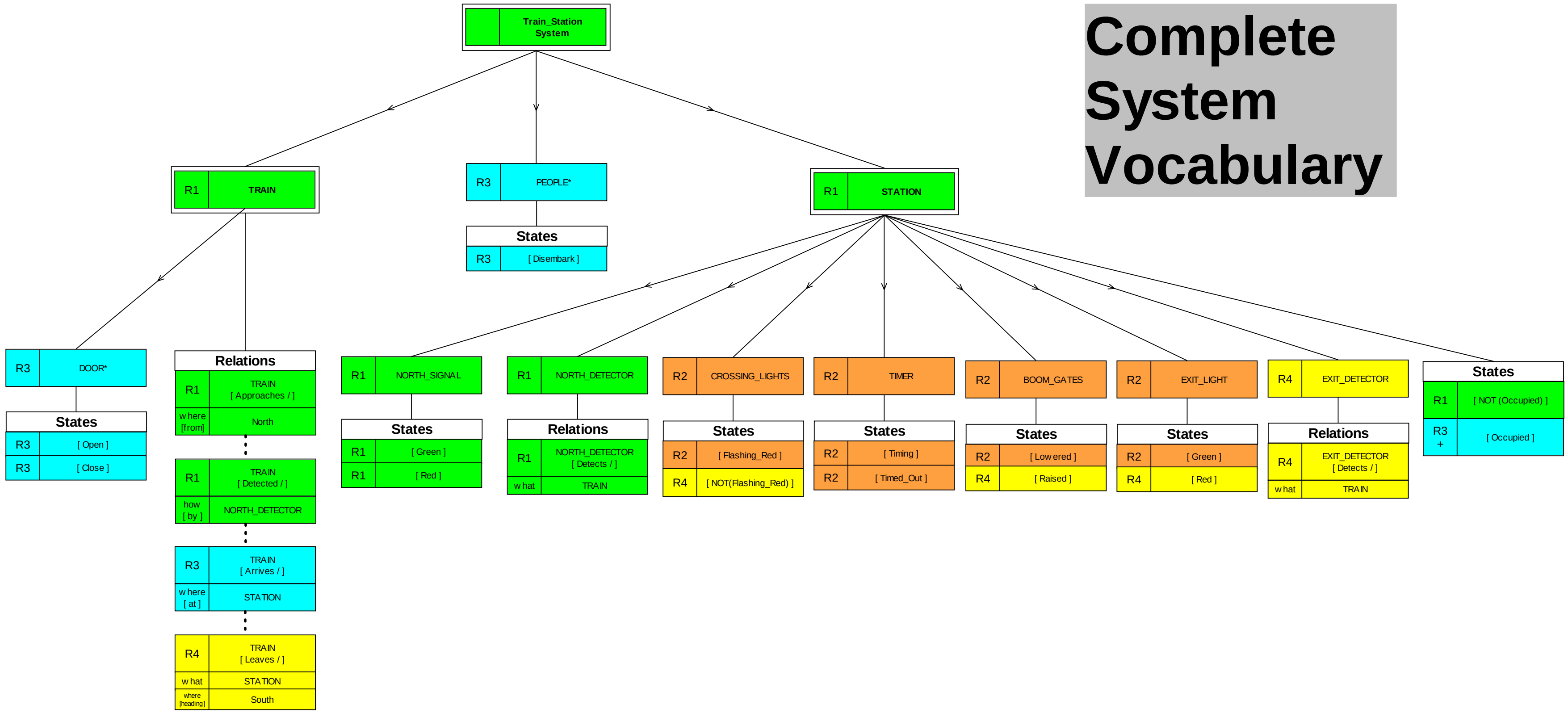
Composition Tree – Station System



Component Hierarchy

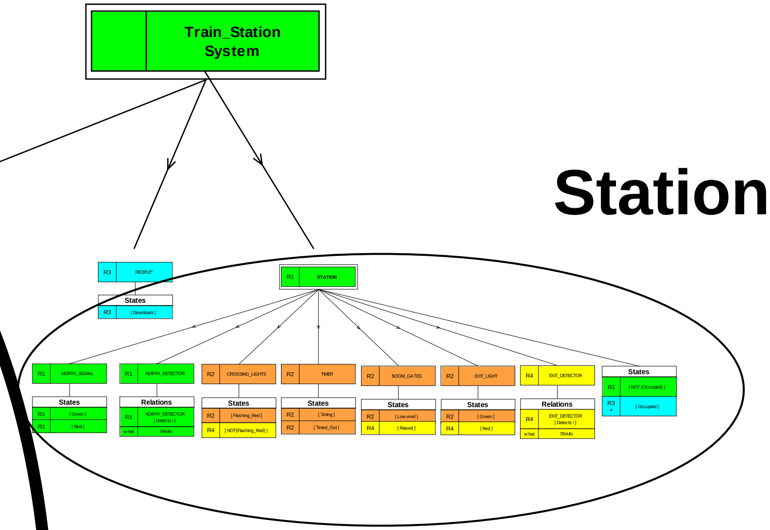
System-of-Systems Integrated View

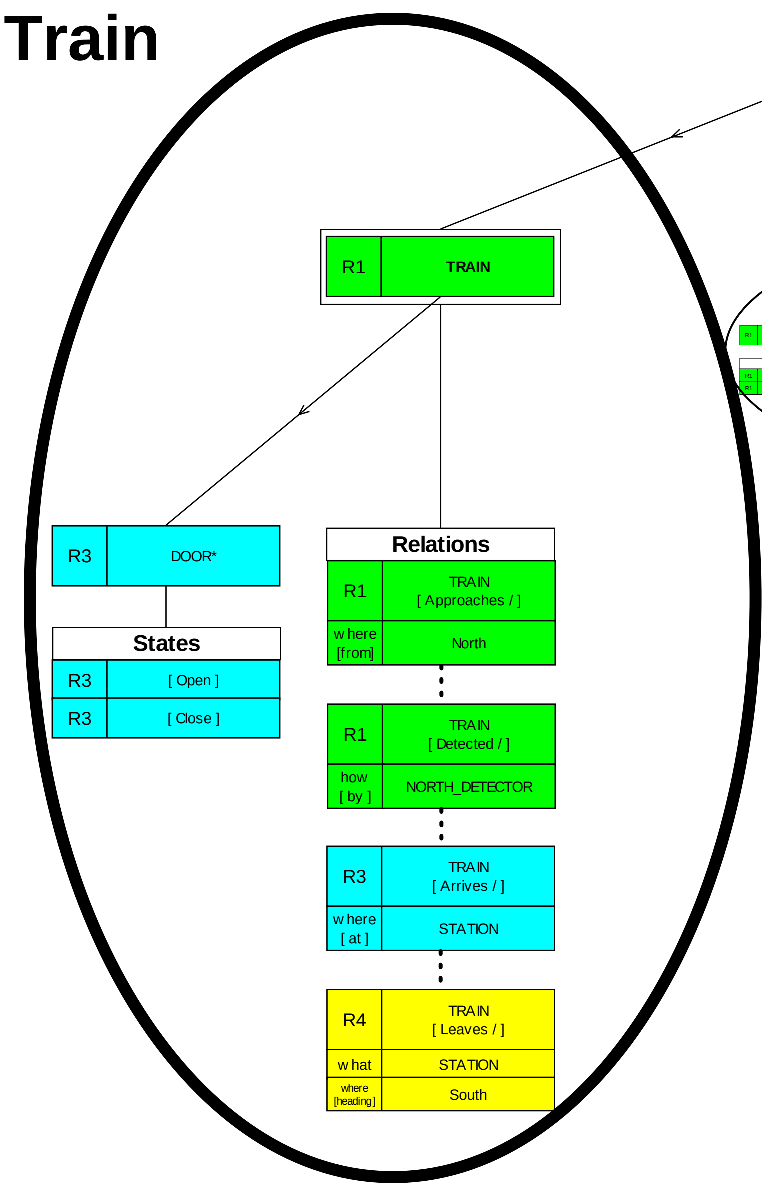
Composition Tree – Station System



Complete System Vocabulary

Composition Tree – Station System





All Information about a single component is in one place

Composition Trees

- Provide an integrated view of data requirements
- Provide integrated knowledge of each component
- Provide a systematic way of finding many types of defects
- Approach repeatability of construction
- Provide information that supports subsequent steps
- Provide an important perspective of the size/dimensionality of the a large system
- Provide vital information that supports understanding and subsequent maintenance of the system
- Provide information that can be easily and usefully refined during later stages of development
- Identify important system architecture information
- Serve to construct the vocabulary of a system

Earlier We Saw

Text → **Behavior Trees**

Now We Want to Consider

Text → **Composition Trees**

Composition Tree

Requirements Translation

+

Integration

Creating an Integrated Compositonal View From Requirements

Example – Train Station System

TRAIN-STATION PROBLEM (Sherwood Station)

Develop a system to model the behavior of a Train-Station. You need to model a train entering the station from the north and then leaving the station to the south. A crossing with boom gates and flashing red lights is located just south of the station. There is a signal to the north of the station that only allows a train to enter when the station is not occupied, that is, when the north signal is green. There is also an exit signal light that ensures the train can only leave the station when the boom gates are down. There is also a north detector that can detect the train approaching the station region from the north. And, there is an exit detector that detects when a train leaves to the south.

1. Initially the station is not occupied. The north signal turns green whenever the station is not occupied. Whenever the north signal is green a train may approach from the north. When approaching from the north a train is detected, by the north detector, which causes the north signal to turn red.

2. When the north detector detects a train it causes the crossing lights to start flashing red. At the same time, a timer starts timing and when it times out it causes the boom gates to be lowered after which the exit light turns green.

3. After the train is detected the north detector, it subsequently arrives at the station, the doors open, the people disembark, and then the doors close.

4. After the doors close the train may leave the station only when and if the exit light is green. When the train leaves the station, heading south, it is detected by the exit detector which means the station is again not occupied. This causes the north signal to turn green and the exit light to turn red. When the exit detector detects the train leaving, it also causes the boom gates to be raised and then the crossing lights to stop flashing red.

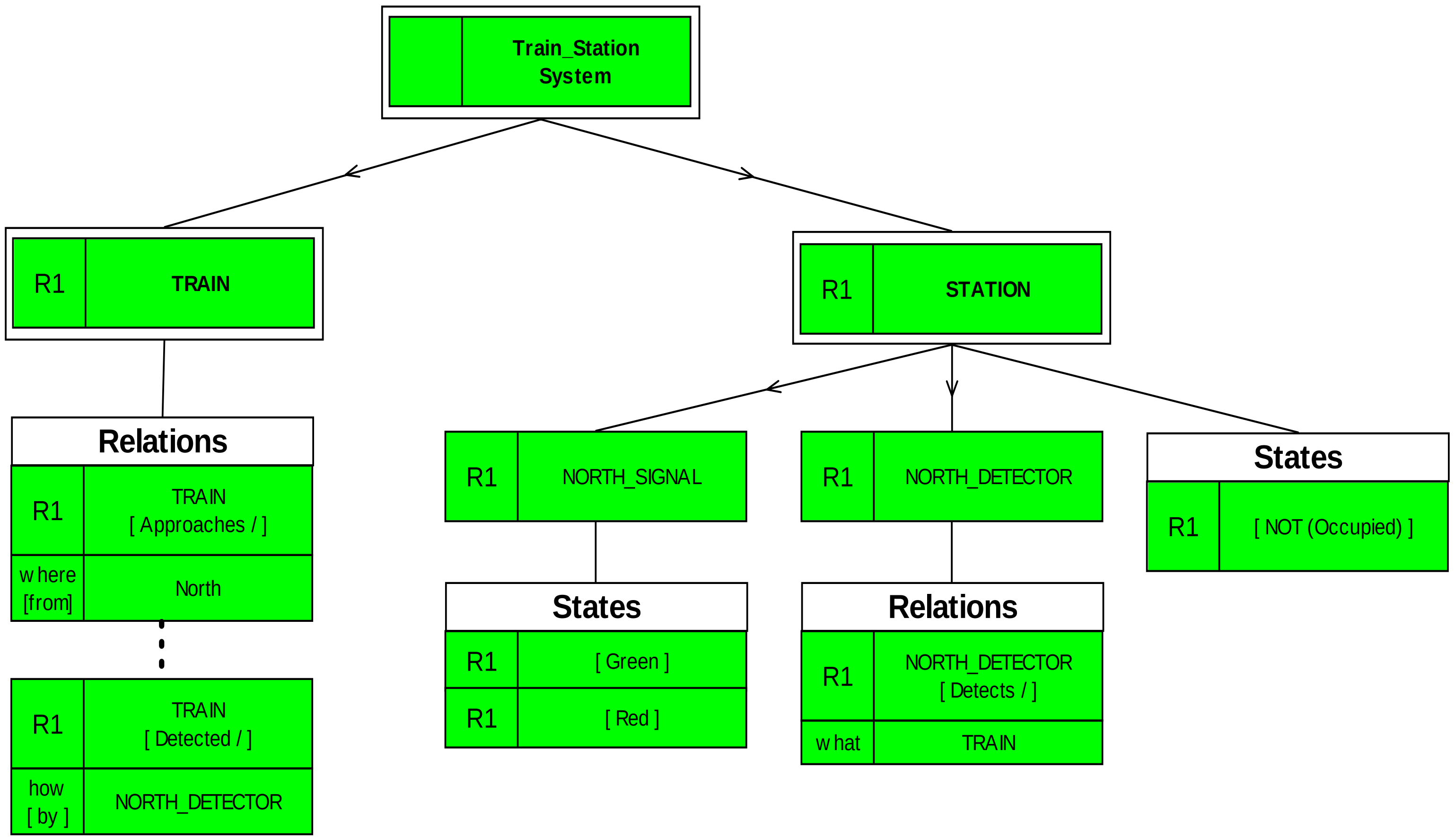
For the purposes of the exercise ignore trains approaching the station from the south. This additional requirement can be integrated later as a separate exercise. Also ignore situations where the train does not stop at the station - this too requires some refinements to the design.

Composition Tree

REQUIREMENT-R1

Initially the station is not occupied. The north signal turns green whenever the station is not occupied. Whenever the north signal is green a train may approach from the north. When approaching from the north, a train is detected by the north detector, which causes the north signal to turn red.

Composition Tree – R1



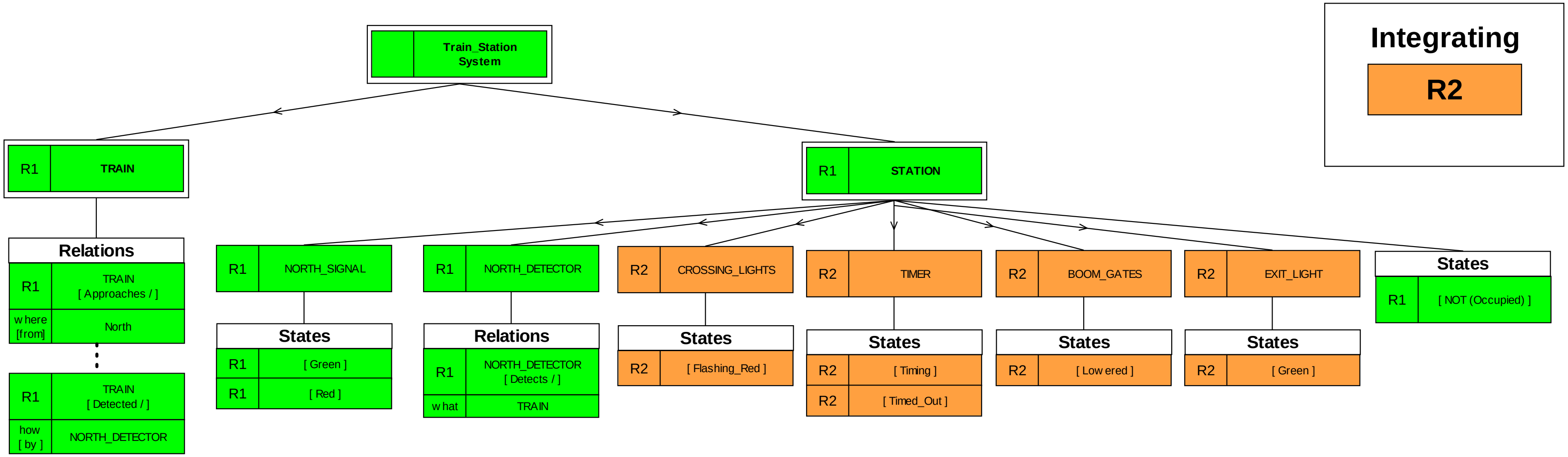
Requirements Translation - One at a time

Composition Tree – Integrating R2

REQUIREMENT-R2

When the north detector detects a train it causes the crossing lights to start flashing red. At the same time a timer starts timing and when it times out, it causes the boom gates to be lowered, after which the exit light turns green.

Composition Tree – Integrating R2



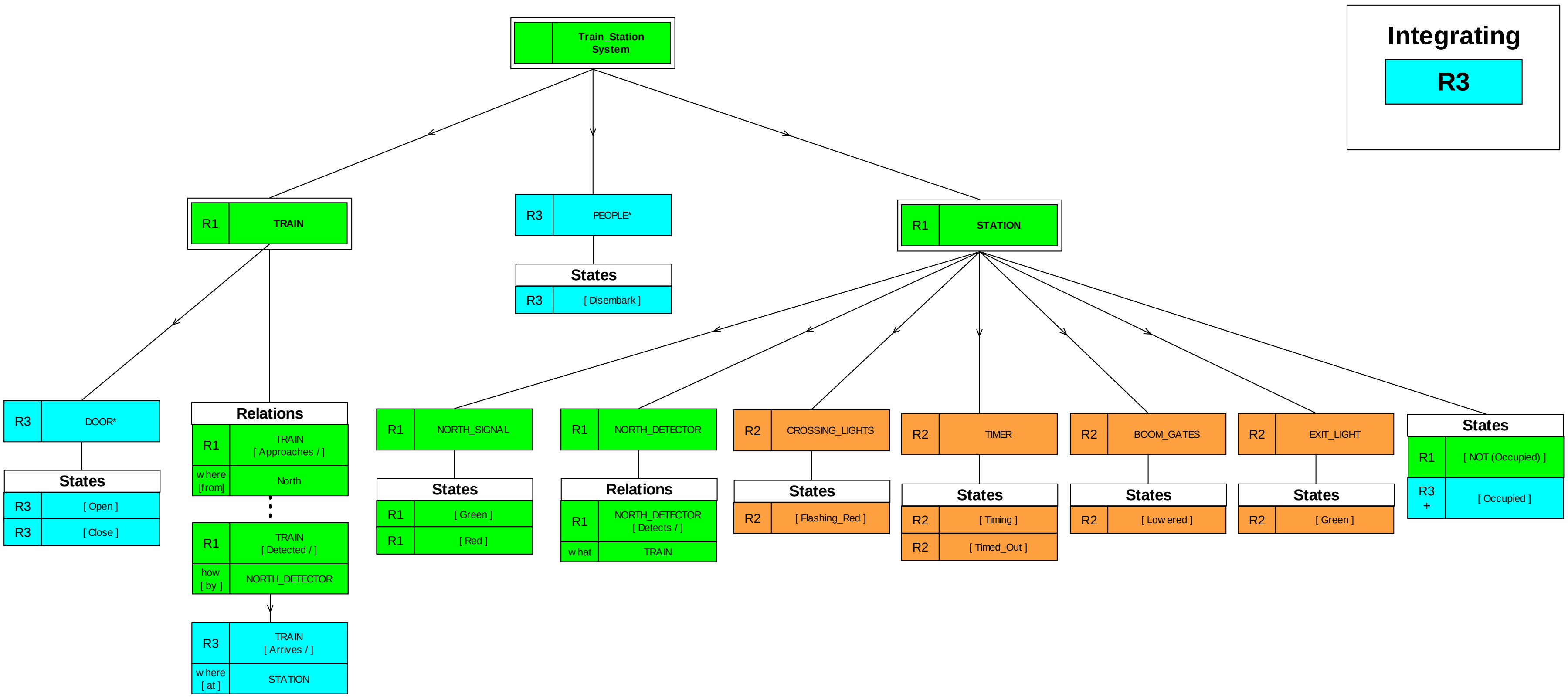
Requirements Translation **PLUS** Integration

Composition Tree – Integrating R3

REQUIREMENT-R3

After the train is detected by the north detector, it subsequently arrives at the station, the doors open, the people disembark, and then the doors close.

Composition Tree – Integrating R3



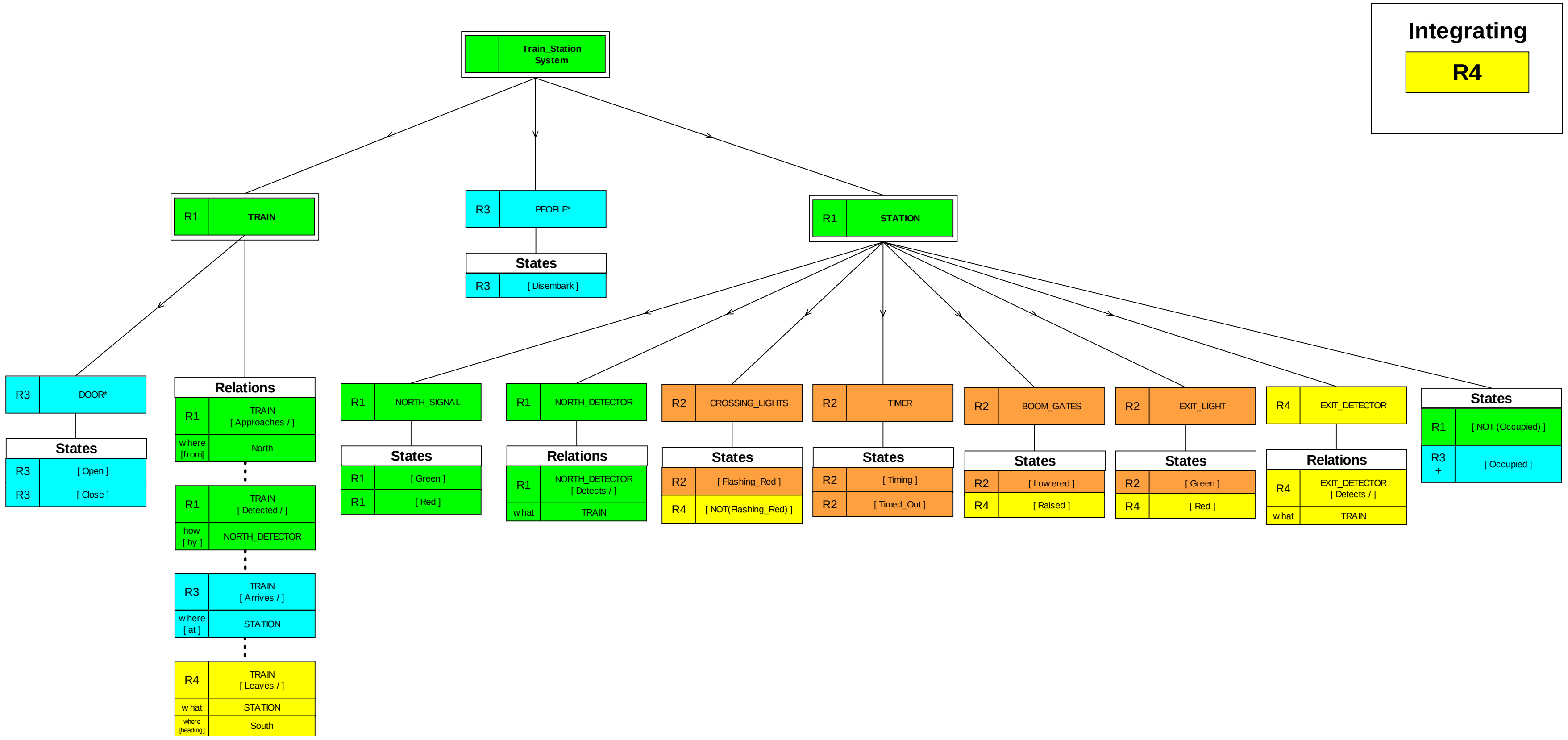
Requirements Translation **PLUS** Integration

Composition Tree – Integrating R4

REQUIREMENT-R4

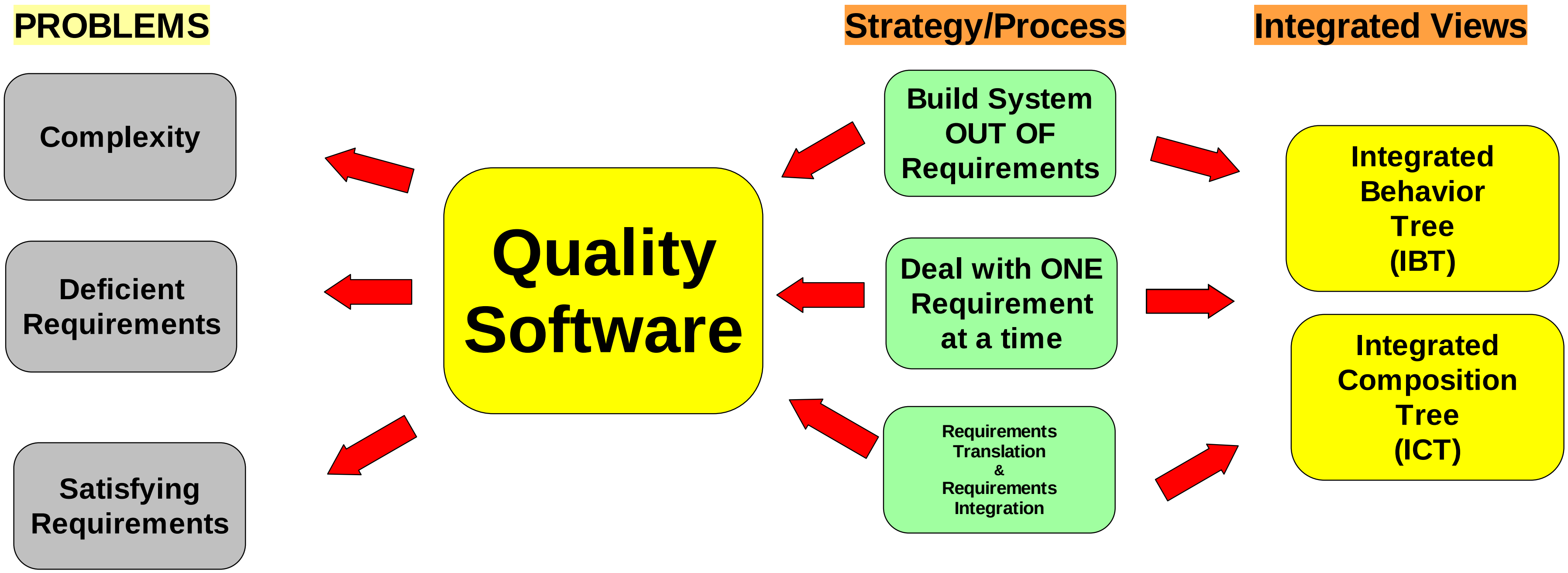
After the doors close the train may leave the station provided the exit light is green. When the train leaves the station, heading south, it is detected by the exit detector, which means the station is again not occupied. This causes the north signal to turn green and the exit light to turn red. When the exit detector detects the train , it also causes the boom gates to be raised and then the crossing lights to stop flashing red.

Composition Tree – Integrating R4

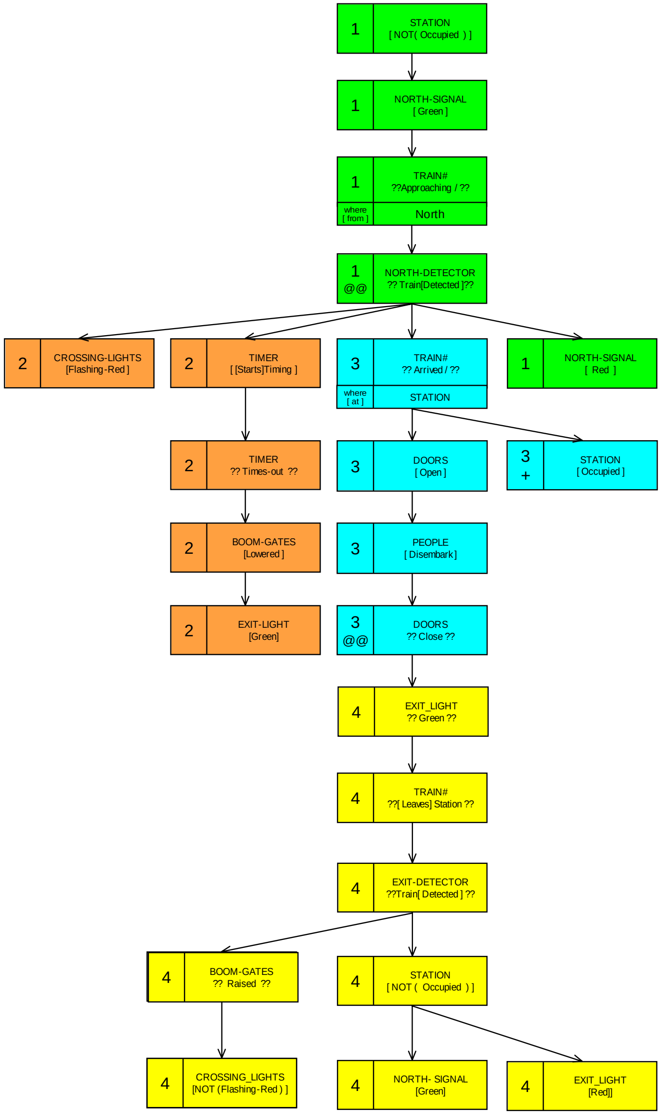


TWO
Integrated Views

Two Integrated Views



Integrated Behavior Tree - IBT

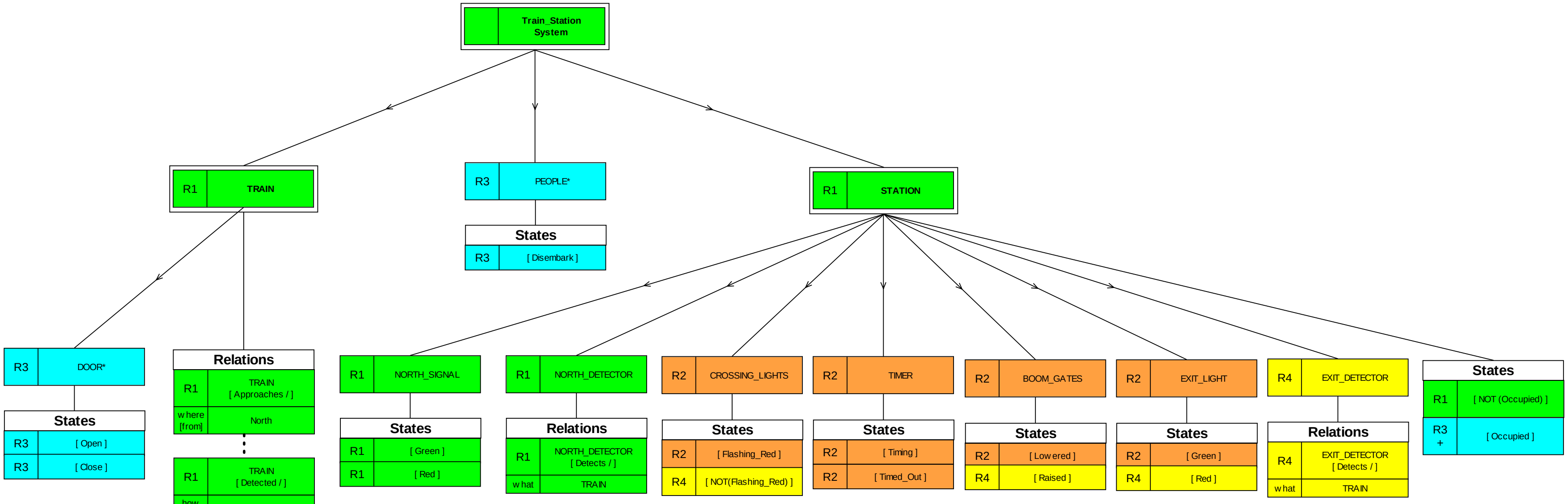


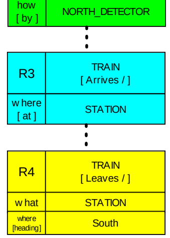
Benefits of Behavior Integration

- See system behavior as a whole
- Integration - detects requirements defects
- Can refine => Specification => Design
- Model-checking, simulation, code-gen.
- Aids component design & implementation

First Integrated View

Integrated Composition Tree - ICT





Benefits of Composition Integration

- All information about a component - is in ONE place
- Aids component design and implementation
- Integration - Helps detect inconsistent information
- Provides complete system vocabulary – all in context

Second Integrated View

Where We Are Up To

Strategy/Process

- Build System OUT OF Requirements
- Deal with ONE Requirement at a time
- Requirements Translation & Requirements Integration

Integrated Views

- Integrated Behavior Tree (IBT) ✓
- Integrated Composition Tree (ICT) ✓

Work Products

- Component Behavior
- Components Interactions
- Integration Specification

- System Vocabulary
- Components' Compositions
- Components Interfaces
- System-of-Systems Hierarchy

Two Integrated Views ✓

Where Are We Up To

Scale & Complexity	✓
Build Right System	✓
Imperfect Knowledge	✓
Productivity	×

Tackling
Team Productivity

Development
By
Teams

Collaborative Editing - Advantages

- Team members translate subsets of requirements.
- Integrated Composition Tree provides strict progressive vocabulary consistency.
- Each team member sees **<u>dynamically</u>** how the work of others affects their work.
- Practical, transparent way to combine the work of individual team members.
- Reduces project team communication overhead.

Potential for significant productivity gains

Microwave Oven – Functional Requirements†

R1. There is a single control button available for the user of the oven. If the oven door is closed and you push the button, the oven will cook (that is, energize the power-tube) for 1 minute.
R2. If you push the button at any time when the oven is cooking, you get an additional minute of cooking time.
R3. Pushing the button when the door is open has no effect.
R4. There is a light inside the oven. Any time the oven is cooking, the light must be turned on. Any time the door is open, the light must be on.
R5. You can stop the cooking by opening the door.
R6. If you close the door, the light goes out. This is the normal configuration when someone has just placed food inside the oven but has not yet pushed the control button.
R7. If the oven times out, it turns off both the power tube and the light. It then emits a warning beep to tell you the food is ready.

Finding
Alias Defects

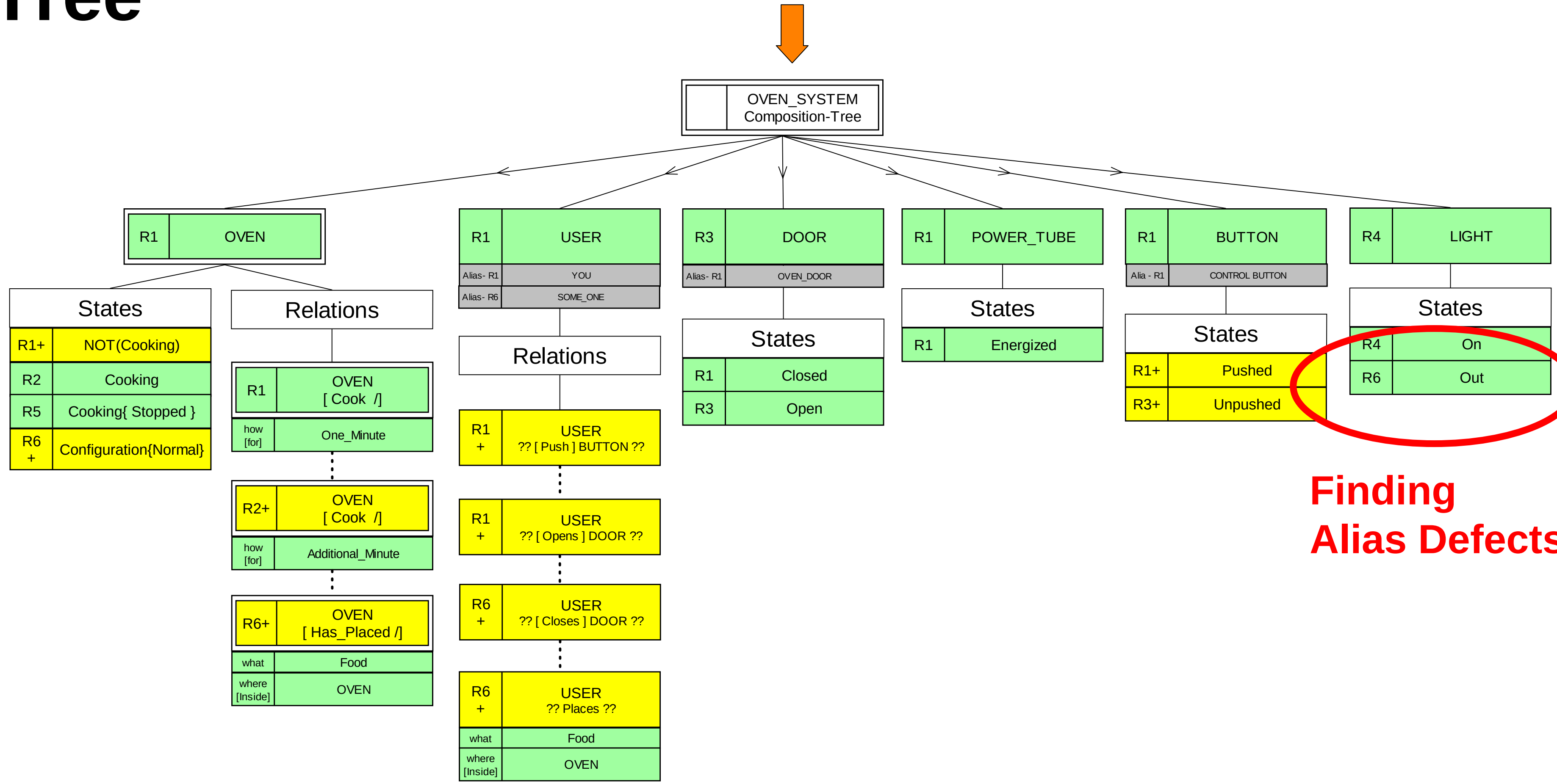
†After Shlaer and Mellor, ***Object Life Cycles, p.36***

Collaborative Editing - ICT

Composition Tree

Requirement-R6
If you close the door the light goes out. This is the normal configuration when someone has just placed food inside the oven but has not yet pushed the control button.

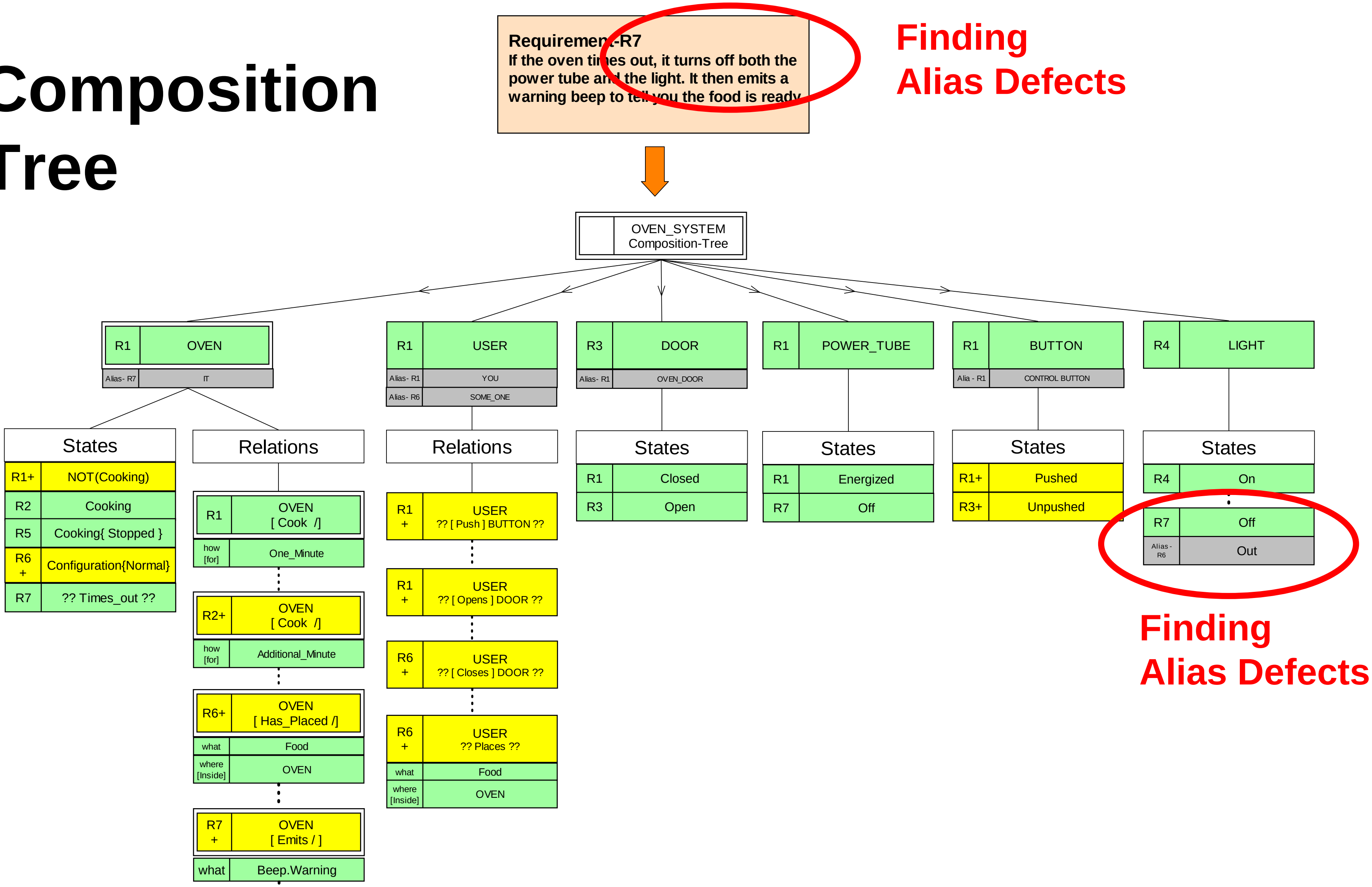
Finding Alias Defects



Finding Alias Defects

Collaborative Editing - ICT

Composition Tree



Finding alias defects

Work

With Industry

Work With Industry

- Behavior Engineering trials on a series of large projects with one large company consistently found 10 – 15% of requirements analyzed contained significant defects not found by their review processes.

- Company is a CMMi company with mature processes.

- Similar statistics on projects for other large companies and organizations

Work With Industry

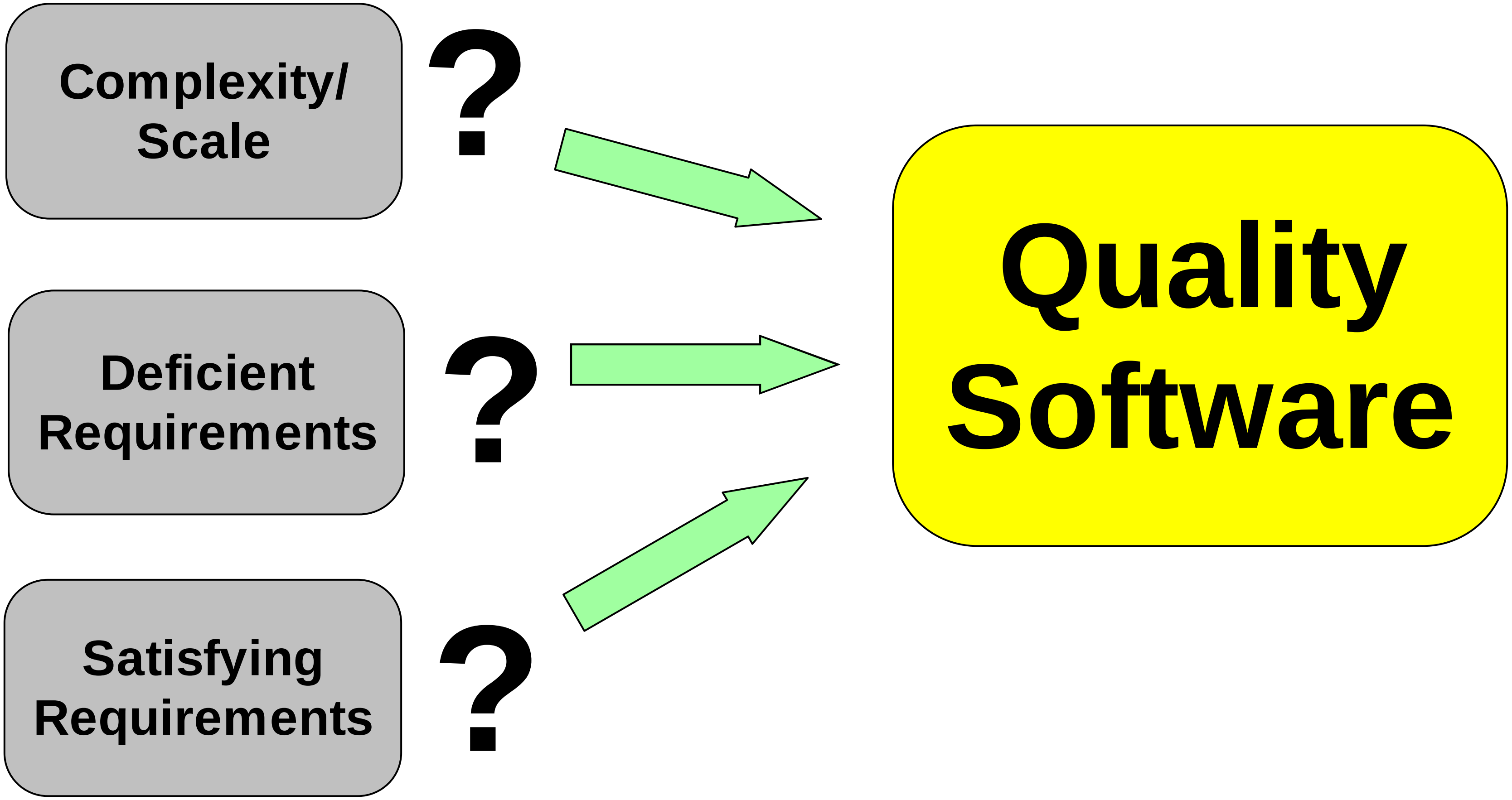
The following table contains statistics on recent projects where we have applied the method.

	Recent Project- RP		Last 3 Projects – (RP excluded)		
			Total	**Ave**	
Number of Pages Analysed:	101pages		265	88.33pages	
Number of Requirements Analysed :	920requirements		3142	1047.33requirements	
Major Defects Only	128defects		412	137.33defects	
Incompleteness	73	57.03%	260	86.67	63.11%
Inconsistency	3	2.34%	30	10.00	7.28%
Ambiguity	19	14.84%	93	31.00	22.57%
Redundancy	31	24.22%	13	4.33	3.16%
Inaccuracy	2	1.56%	16	5.33	3.88%
Number of Queries:	7queries		98	32.67queries	
Effort (Includes reporting, analysis, modeling)	94Person-hours		325	108.33Person-hours	

What the results show is that the Behavior Engineering method consistently finds 130 major defects per 1000 of requirements **after** normal reviews and correction have been carried out. In addition the integrated work products constructed to detect defects can subsequently be corrected and refined to create an executable design.

Where have we got to?

Threats to Producing Quality Software



These Problems are all interdependent

Behavior Engineering

Tackling Complexity Head-on

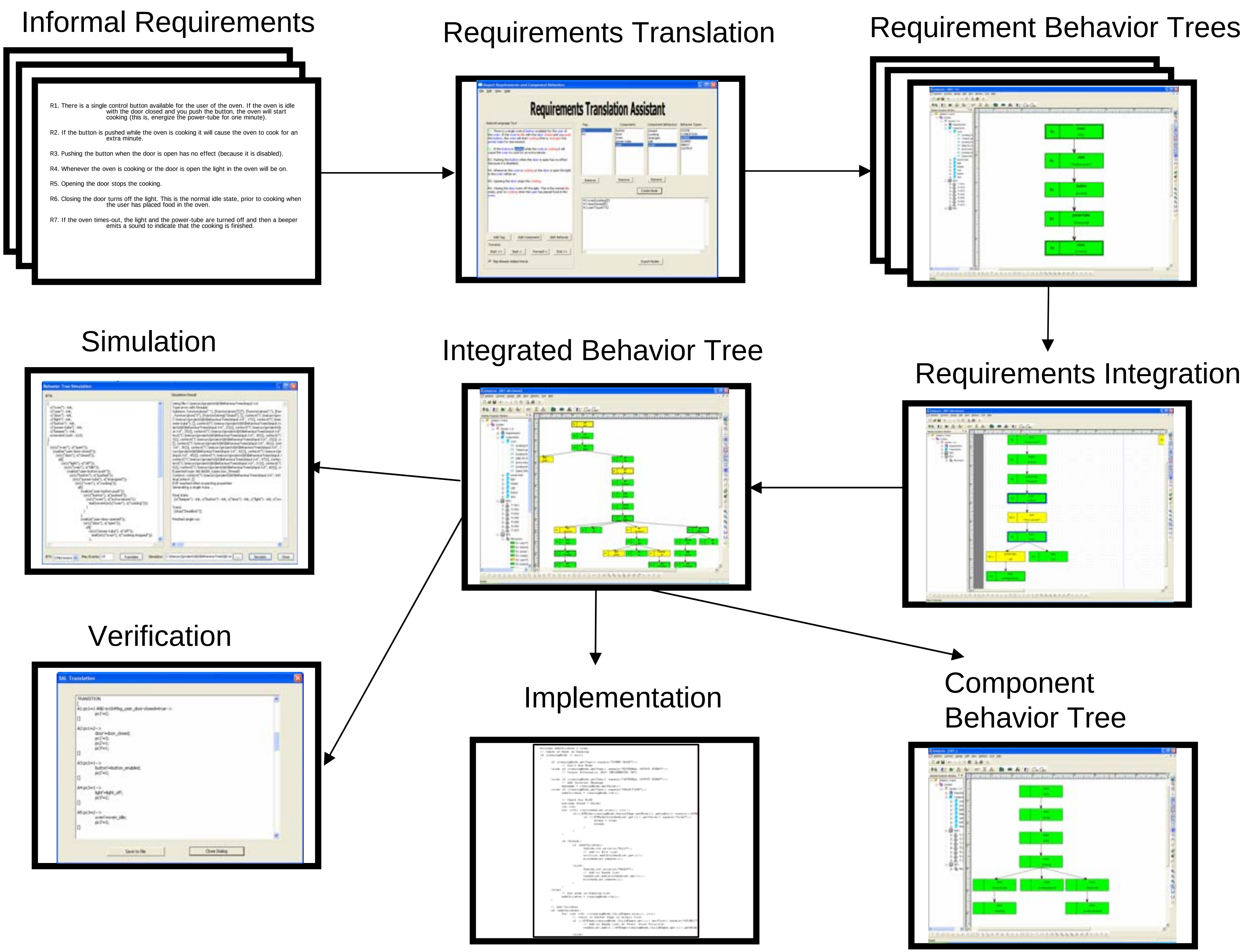
Behavior Engineering

Tackling Deficient Requirements Head-on

Behavior Engineering

Tackling Verification & Validation Head-on

Building Dependable Systems



Simple, Scaleable Development

- Accuracy – individual requirements translation
- Validation – preserve original vocabulary
- Complexity – deal one requirement at a time
- Defects – rigorous translation, integration, MC
- Comprehending – requirements integration
- Dividing up the work – single requirement focus

Audience – Acquiring New Knowledge

"There are two ways of acquiring knowledge … Argument reaches a conclusion and compels us to admit it, but it neither makes us certain nor so annihilates doubt that the mind rests calm in the intuition of truth, unless it finds this certitude by way of experience"

- Roger Bacon, 1268 AD

… and more information

www.behaviorengineering.org
www.accs.edu.au

Acknowledgement

I would like acknowledge the contribution of my colleagues at the ARC Centre for Complex Systems at University of Queensland and my colleagues and students at Griffith University who have contributed to this work. I would also like to thank the many people in Industry and academia who have supported and encouraged me in progressing this work over the last seven years.

***“If you keep doing what you have always done, you will keep getting what you have always got”*.**

– W. Edwards Deming

Take-home message

“I believe that failure is less frequently attributable to either insufficiency of means or impatience of labour than to a confused understanding of the thing actually to be done.”

John Ruskin