Two Synthesis Applications:
Boeing/Navy/Stars Flight Training Systems and Rockwell Command & Control Systems
Best Available Copy
TWO SYNTHESIS APPLICATIONS:

BOEING/NAVY/STARS FLIGHT TRAINING SYSTEMS

K.C. King

AND ROCKWELL COMMAND & CONTROL SYSTEMS

Jerri Turner-Harris

SPC-94035-CMC

VERSION 01.00.00

MAY 1994

This material is based in part upon work sponsored by the Advanced Research Projects Agency under Grant # MDA972-92-J-1018. The content does not necessarily reflect the position or the policy of the U.S. Government, and no official endorsement should be inferred.

This document accompanies a videotape of the same presentation recorded live at the Software Productivity Consortium in March 1994. It is recommended that the videotape be viewed with these viewgraphs at hand.

Produced by the
SOFTWARE PRODUCTIVITY CONSORTIUM
under contract to the
VIRGINIA CENTER OF EXCELLENCE
FOR SOFTWARE REUSE AND TECHNOLOGY TRANSFER
SPC Building
2214 Rock Hill Road
Herndon, Virginia 22070
ABSTRACT

TWO SYNTHESIS APPLICATIONS:
BOEING/NAVY/STARS FLIGHT TRAINING SYSTEMS
AND ROCKWELL COMMAND & CONTROL SYSTEMS

K.C. King, Boeing STARS Demonstration Project Manager
Jerri Turner-Harris, Rockwell’s Command and Control Systems Division

K.C. King presents the Boeing/Navy/STARS demonstration of the Consortium’s Synthesis methodology for software reuse on Navy Flight Training Systems. Synthesis is the foundation for an approach being used by Boeing and a U.S. Navy/STARS team to develop software for the Navy’s T-34C flight simulator trainer. A major reuse strategy of the program is the use of a “two life-cycle model”, requiring that the traditional application development life cycle be “front-ended” with a separate, but coordinated, life cycle that creates process-driven software assets for a defined product line (or family) of systems. The mastering of variations in requirements between similar systems is at the heart of Synthesis. Mr. King describes how they have used Synthesis in a comprehensive, “leveraged” mode, fully deploying the Synthesis methodology throughout the program.

Jerri Turner-Harris of Rockwell’s CCSD discusses their use of Synthesis to create a “domain engineering” approach to developing interprocessor communication and message handling systems for commercial clients. Ms. Turner-Harris explains how Rockwell is implementing Synthesis in their communications domain to eventually allow them to develop new systems in a fraction of the time currently required.

This video is intended for systems and software engineering development lead engineers, project managers, and division managers working in the area of system and software engineering. Viewers will benefit by gaining insight into the synthesis process and the experiences of others who are successfully applying synthesis.
ARPA STARS PROGRAM
Boeing/Navy Demonstration Project
Navy Flight Training Systems

Presented by
K. C. King
Boeing STARS Demo Program Manager
<table>
<thead>
<tr>
<th>Were processes documented?</th>
<th>Were metrics collected?</th>
<th>Were the requirements developed in the time allocated?</th>
<th>Have the functional requirements for Nav/Com been met?</th>
<th>Have the AE processes been followed?</th>
<th>Has the AE process been enacted in the SEE?</th>
<th>Have the required work products been developed?</th>
<th>Have lessons learned been documented?</th>
</tr>
</thead>
</table>

**Success Criteria (Explicit)**
## Success Criteria (Implicit)

Is the team ready for the Performance Phase?

<table>
<thead>
<tr>
<th>Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>The team has a complete set of processes</td>
</tr>
<tr>
<td>The team has confidence in the effectiveness of those processes and the capability to perform them</td>
</tr>
<tr>
<td>There is effective automated support for these processes</td>
</tr>
<tr>
<td>The team understands remaining risks and has a plan to control their impact</td>
</tr>
<tr>
<td>The team has a realistic, agreed to, and a visible plan to accomplish the demo</td>
</tr>
<tr>
<td>The team has the critical resources and a credible plan for acquiring additional resources</td>
</tr>
<tr>
<td>Senior leadership is committed to giving mega programming a fair test</td>
</tr>
</tbody>
</table>
AVTS Pilot Demonstration Project
Document Tree

Megaprogramming Process

Software Design Methodology

Organizational Processes

Domain Engineering Work Products

Application Engineering Work Products

Reuse-Driven Software Processes Guidebook (Synthesis - VCOE)

Synthesis Guidelines

Decision Model Guidelines

DARTS Documentation

Functional Analysis

Subsystem Controller Entry Points

Process Requirements Paradigm

Algorithm Prototyping

Design Review

Design Guidelines

Generation Design Updates

Structural Analysis

Structural Model

For Navigation

Domain Synopsis Process Requirements

Problem Management System Domain Assumptions Product Architecture

Domain Management Process Domain Glossary Component Design

For Navigation

Domain Status Generation Design

Decision Model Component Implementation

Product Requirements Generation Implementation

For Navigation

Instance Decision Vector

Instance Requirements

Instance Adapted Code

Instance Acceptance Test

Instance Parts Drawing

* See Notebook entitled:
Supporting Material for Process Guidelines
Domain Management

Purpose: Responsible for managing business area resources to achieve design business objectives

Product: Domain Plan

- Domain Evolution Plan - Long Term
- Domain Increment Plan - Near Term
- Practices and Procedures

Lessons Learned:

- Formal Domain Plan is essential
Navy/STARS Readiness Review

Domain Definition

• Concepts:
  - Description of the scope of the domain
  - Definition of the terminologies
  - Identification of commonalities and variabilities
  - Determine technical maturity of the domain
  - Determine exclusions within the domain

• Work products examples
  - Synopsis ✓
  - Glossary ✓
  - Assumptions
  - Status ✓

• Success indications
  - Work products developed
  - Lessons learned documented
    - Define variability early
    - Standardize terminologies (Synthesis/Darts)
Domain Analysis

- **Concepts:**
  - Business Case
  - Domain expertise
  - Process and product development for use by an Application Engineer

- **Examples Work Products:**
  - Domain Definition
  - Domain Specification

- **Success Indicators**
  - All success criteria were met
NAVIGATION/COMMUNICATION ASSUMPTIONS

COMMONALITIES

2.5 Self test for TACAN radios have the following commonalities:
   o Each has a set of discrete events that occur over a finite amount of time.
   o There is a way to manually initiate self test.
   o Self test results in TACAN outputs being driven to predefined values.

Justification: Based on analysis of self test characteristics using 3 aircraft as examples. (T34, T44, T45).

VARIABILITIES

2.1.10 Self Test characteristics vary for each type of radio in the domain.
2.1.10.1 The ways that self test is initiated varies for each type of radio.
2.1.10.2 The duration of self test varies for each type of radio.
2.1.10.3 The characteristics that are exhibited during self test varies for each radio.
2.1.10.4 The way self test is terminated varies for each type of radio.
2.1.10.5 The interaction of TACAN/VOR self test with other systems varies for given cockpit configurations.
2.1.10.6 Self Test may or may not exist for TACAN/VOR.

Justification: For real aircraft in the domain, detailed self test characteristics will vary.

EXCLUSIONS

DOMAIN ASSUMPTIONS
Decision Model

• Concepts:
  - Basis (foundation) for the Application Engineering process
  - Distinguishes leverage reuse through identification variabilities in the form of questions
  - Basis for "tailoring" generics to specifics

• Examples work products within the Decision Model:
  - Decision Tables - a set of decisions representing the variability assumptions
  - Decision Group - a set of related decisions
  - Decision Constraints - Corresponds to the number of possible decisions.

• Success indications:
  - Work products were produced
  - Functionality was addressed
  - Lessons learned
    • Domain experts are essential
    • Design of Decision Model should be driven by how best to represent variability, not presentation format
    • Decision Groups should be kept small
<table>
<thead>
<tr>
<th>Decision Variable</th>
<th>Structural Constraint</th>
<th>DC Ref</th>
<th>VA Ref</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change_In_Channel_Terminate_TA-CAN_Self_Test</td>
<td>Exactly One (Yes, No)</td>
<td>VA_2.1.10.4</td>
<td>For TACAN Self Test there are several ways to terminate the self test.</td>
<td></td>
</tr>
<tr>
<td>Components_Affected_By_Tacan_Self_Test</td>
<td>Zero or More [VOR, INS, RNAV, ADF]</td>
<td>VA_2.1.10.5</td>
<td>For TACAN there may be other systems that react to a TACAN self test signal</td>
<td></td>
</tr>
<tr>
<td>TACAN_Self_Test_Initiated</td>
<td>One or More [Push and Hold, Push and Release, Power On, Other]</td>
<td>VA_2.1.10.1</td>
<td>For initiating self test there are several options. These options are based on the type of control unit under consideration.</td>
<td></td>
</tr>
<tr>
<td>TACAN_Button_Pressed_During_Self_Test</td>
<td>Exactly One [Restart, Terminate, Nothing]</td>
<td>DC_2</td>
<td>VA_2.1.10.1</td>
<td>For TACAN self test initiated by a push and type button, there is variability in the effect of pressing the button again during self test.</td>
</tr>
<tr>
<td>TACAN_Self_Test_Phases</td>
<td>Integer in Range [0,25]</td>
<td>VA_2.1.10.2</td>
<td>In a TACAN radio, there are a variable number of discrete events that occur during self test. Each phase is defined by a change in the characteristics of self test.</td>
<td></td>
</tr>
</tbody>
</table>

**DECISION MODELS**
Product Requirements

- **Concepts:**
  - Describes the product family scope & behavior
  - Describes the purpose and objective of the product family
  - Describes the relationship within the product family
  - Describes the limits of the product family
  - Products and Process Requirements can be done concurrently (and was)

- **Example work products**
  - Product Requirements Specification

- **Success indications:**
  - Functional requirements were developed
  - Requirements were developed in the time allocated
  - Work products were developed
Navigation/Communication Product Requirements

1. INTRODUCTION
2. REQUIREMENTS.
2.1 Segment Definition.
2.2 Characteristics.
2.2.1 Performance Characteristics.
2.2.1.1 Segment Modes and States.
2.2.1.2 Nav/Comm Segment Functions. The following functions shall be accomplished by the Nav/Comm segment.
2.2.1.2.1 Nav/Comm Segment Support Function.

\[\text{if Radio Navigation Allds then}\]
\[\text{if VHF Nav System then}\]
\[\text{if VHF Nav Self Test Required then}\]
There shall be a self test function for the VHF Nav system.
\[\text{if VOR Self Test Initiated (Push and Hold) then}\]
The VOR self test shall simulate the characteristics of self test by pressing and holding the self test switch.
\[\text{end if}\]
\[\text{end if}\]
\[\text{end if}\]
\[\text{if TACAN System then}\]
\[\text{if TACAN Self Test Required then}\]
There shall be a self test function for the TACAN system.
\[\text{if TACAN Self Test Initiated (Push and Hold) then}\]
The TACAN self test shall be initiated by a press and hold button.
\[\text{end if}\]
\[\text{end if}\]
\[\text{end if}\]
Process Requirements

- Concepts:
  - Describes the Application Engineering process
  - Describes the procedures for identifying reuse

- Examples work product:
  - Process Specification
  - Application Modeling Notation Specification

- Success indications:
  - Work products were developed
  - Application Engineering process was documented
TACAN DECISION GROUP

NC.RA.TN

TACAN Decision Group (NC.RA.TN)

What TACAN Crew Locations Are Required?

Pilot Copilot

Is a TACAN Self Test Required?

Is TACAN Secondary Test Required?

Are TACAN Degraded Modes Of Operation Required?

TACAN Modes Decision Group

TACAN Self Test Decision Group

TACAN Secondary Test Decision Group

TACAN Degraded Modes Of Operation Decision Group

TACAN Control Head Decision Group

NC.RA.TN.TMO

NC.RA.TN.TST

NC.RA.TN.TSCT

NC.RA.TN.TDG

NC.RA.TN.TCH

What is the TACAN Make Model?

TCN-40 AN/ARN-136(V) Other
Product Design

- **Concepts:**
  - Specifies the design of the product family
  - Development of design methodologies

- **Example work products:**
  - Product Architecture
  - Component Design
  - Generation Design

- **Success Indications**
  - Work products have been developed
  - Lessons learned have been documented
    - Design methodologies are required as entrance criteria
    - Design should allow for feedback loop between component design and architecture
Architecture

• Concepts:
  - Adopted a mature architecture for the process (DARTS)
  - Supported by design methods (flow, dependency & object) developed by the team
  - AVTS (DARTS message handler) central to overall structure

• Example work products:
  - Product Architecture

• Success indication:
  - Work products have been developed
  - Lessons learned have been documented
    • Product Architecture must provide necessary levels of encapsulation
    • Product Architecture can be fine grained
    • Architecture does not necessarily match the Decision Model
Domain Architecture for Reusable Training Systems (DARTS)
Navy/STARS Readiness Review

Component Design

• Concept:
  - Component design is similar to traditional design
  - Component design defines the interfaces

• Example work products
  - Adaptation Specification
  - Interface Specification

• Success indications:
  - Work products were developed
  - Lessons Learned were documented
    - Component Design effects product architecture and vice versa
self_test_active = true  — set self test to active
endif
endif

<else if P_TAC_ST_PUSH_AND_HOLD> — if it's a press and hold push button.
  if self_test_activate = true  — Self test button pressed then
    self_Test_Activate = true  — set self test to active
  endif
<else>
  null;  — none or other manual self test initiation
<endif>

— process self test
if self test active
  increment self_test_timer;
  — check for premature termination
  <if P_TAC_TERM_WITH_CHAN then> — if TACAN self test terminated by change in
    if change_in_channel then
      terminate_self_test
    endif
  <endif>
  <if other termination condition then>
    Terminate_self_test
  <endif>
  — now determine the phase
  phase_found = false  — initialize a flag for search loop.
loop for each phase
  if timer < phase end time then  — for each phase of time that's defined in self test.
    phase = loop index;
    phase_found = true  — set the phase found flag
    exit loop;
  endloop  — set the phase found flag
  if phase_found = false  — if all phases have run to completion.
    terminate_self_test;
  endif
endif:  — end process self test logic

procedure Terminate_Self_Test;
Generation Design

- **Concepts:**
  - Specifies the mapping that will produce Application Engineering work products
  - Generated Design is the adaptation of the Component Design
- **Example mappings**
  - *Architecture Mapping* - Decision Model-to-Architecture
  - *Component Mapping* - How each component of a work product is to be produce
  - *Decision Mapping* - Relations between Decision Model and instantiation parameters for the Component Design

- **Success indications**
  - Work products were developed
  - Lessons learned were documented
    - Process support is required
| **P INITIATE PUSH_AND_HOLD** | if (Push_and_Hold ∈ loc%_inst%_tst%_Test_Initiated)  
| then TRUE  
| else FALSE |
| **P INITIATE_REMOTE** | if (Remote ∈ loc%_inst%_tst%_Test_Initiated)  
| then TRUE  
| else FALSE |
| **P AFFECT_OF_REPRESS_RESTART** | if (%loc%_inst%_Button_Pressed_During_%tst%_Test = Restart)  
| then TRUE  
| else FALSE |
| **P AFFECT_OF_REPRESS_TERMINATE** | if (%loc%_inst%_Button_Pressed_During_%tst%_Test = Terminate)  
| then TRUE  
| else FALSE |
| **P AFFECT_OF_REPRESS_NO_EFFECT** | if (%loc%_inst%_Button_Pressed_During_%tst%_Test = Nothing)  
| then TRUE  
| else FALSE |
| **P_ST_FAIL** | if (Self_Test_Fail ∈ TACAN_Malfunctions)  
| then TRUE  
| else FALSE |

**DECISION MAPPING**
Architecture Map – provides the mapping between the architecture nodes and the instantiation parameters. This defines what architectural nodes are required given an application model (instance of the domain).

NAV/COMM Segment Exec
   If Navigation_Communication_System = Yes

Radio Navigation Aids
   If Radio_Navigation_Aids = Yes

Pilot TACAN Control
   If TACAN_Systems = Yes and {Pilot is an element of TACAN_Crew_Locations}

Pilot TACAN Resolve Power
   If TACAN_Systems = Yes and {Pilot is an element of TACAN_Crew_Locations}

Pilot TACAN Determine Self Test Status
   If Pilot_TACAN_Self_Test_Required = Yes

Pilot TACAN Maintain Station Data
   If TACAN_Systems = Yes and {Pilot is an element of TACAN_Crew_Locations}

Calculate Pilot TACAN Nav Outputs
   If TACAN_Systems = Yes and {Pilot is an element of TACAN_Crew_Locations}

Calculate Pilot TACAN Control Head Outputs
   If TACAN_Systems = Yes and {Pilot is an element of TACAN_Crew_Locations}

Pilot VOR Control
   If VHF_Nav_System = Yes and {Pilot is an element of VOR_Crew_Locations}

Pilot VOR Resolve Power
   If VHF_Nav_System = Yes and {Pilot is an element of VOR_Crew_Locations}

Pilot VOR Determine Self Test Status
   If Pilot_VHF_Nav_Self_Test_Required = Yes

Pilot VOR Maintain Station Data
   If VHF_Nav_System = Yes and {Pilot is an element of VOR_Crew_Locations}

Calculate Pilot VOR Nav Outputs
   If VHF_Nav_System = Yes and {Pilot is an element of VOR_Crew_Locations}

Calculate Pilot VOR Control Head Outputs
   If VHF_Nav_System = Yes and {Pilot is an element of VOR_Crew_Locations}
**COMPONENT MAPPING**

Component Map – provides the mapping between the architectural nodes and the adaptable components that fill the node.

**NAV/COMM Segment Exec**

Use Component: Package, Navigation_Communication_Executive

**Radio Navigation Aids**

Use Component: Package, Radio_Nav_Aids_Subsystem_Controller

**Pilot TACAN Control**

Use Component: Package, TACAN

**Pilot TACAN Resolve Power**

Use Component: Procedure, TACAN_Resolve_Power

**Pilot TACAN Determine Self Test Status**

Use Component: Package, Self_Test

**Pilot TACAN Maintain Station Data**

Use Component: Procedure, Maintain_Station_Data

**Calculate Pilot TACAN Nav Outputs**

Use Component: Procedure, TACAN_Nav_Outputs

**Calculate Pilot TACAN Control Head Outputs**

Use Component: Procedure, TACAN_Control_Head_Outputs

**Pilot VOR Control**

Use Component: Package, VOR

**Pilot VOR Resolve Power**

Use Component: Procedure, VOR_Resolve_Power

**Pilot VOR Determine Self Test Status**

Use Component: Package, Self_Test

**Pilot VOR Maintain Station Data**

Use Component: Procedure, Maintain_Station_Data

**Calculate Pilot VOR Nav Outputs**

Use Component: Procedure, VOR_Nav_Outputs

**Calculate Pilot VOR Control Head Outputs**

Use Component: Procedure, VOR_Control_Head_Outputs
Navy/STARS Readiness Review

Domain Implementation

• **Key activities:**
  - Product Implementation
    • Component Implementation
    • Generation Implementation
  - Process Support Development

• **Success indication:**
  - Work products have been developed
  - Application Engineering process was documented
  - Application Engineering process was developed
  - Application Engineering process exacted in SEE
  - Lessons learned were documented
    • ADA skills are required
Component and Generation Implementation

• Concepts:
  - Implement components with variables as instantiation parameters
  - Create automated processes to:
    • Capture variations as questions
    • Retrieve and adapt components

• Examples:
  - Adaptable components
  - Generation procedures

• Success indications
  - Work products have been developed
  - Application Engineering process has been developed
  - Lessons learned have been documented
    • Generation Implementation should be performed by a process engineer
(deffunction make-TACANSelfTest (?spawninst ?value)
    (if (eq ?TACAN_C_answer_value Yes) then
        (bind ?value Copilot)
        (bind ?nvalues (+ ?nvalues 1))
    )
    (if (eq ?TACAN_P_answer_value Yes) then
        (bind ?value Pilot)
        (bind ?nvalues (+ ?nvalues 1))
    )

    (bind ?newsym (sym-cat ?value "TACSlfTestInit"))
    (bind ?newdesc (str-cat ?value "TACSlfTestInit"))

    (bind ?newnam (symbol-to-instance-name ?newsym))
    (bind ?qnewsym (sym-cat ?newsym "Question"))
    (bind ?qnewnam (symbol-to-instance-name ?qnewsym))

    (make-instance ?newnam of MULTI_DECISION
        (Design_Group_ID TACANSelfTest)
        (Question_Name ?qnewsym)
        (One_Line_Desc ?newdesc)
        (InstanceInstantiatedBy ?spawninst)
        (FurtherQFunction make-buttonpress)
        (Multi_Entry nil))
    (make-instance ?qnewnam of MULTIPLE_CHOICE_QUESTION
        (Decision_Name ?newsym)
        (Question (format nil "How is " ?value "TACAN self test Initiated?")
            (One_Line_Question ?newdesc)
            (Text_Choices Push_And_Hold Power_On Push_And_Release Other)
            (Choices Push_And_Hold Power_on Push_And_Release Other)
            (Lines_Selected 0)
            (Decision_Help_Text (format nil "For initiating self test there are"
                "several options. These options are based on ",
                "the type of control unit under consideration"))

    KAMEL MAKE FUNCTION
DECIDE SCREEN

File Actions

How is Pilot TACAN self test initiated?

- Push_And_Hold
- Power_On

- Push_And_Release
- Other

- Malfunctions: Yes
- Backdoor Interface: No
- Diagnostics/Test: No
- Segment--Occulting: Environment
- Segment--RadarDB/GA: None
- Segment--VisualDB/GA: None
- Segment--SpatialRel: Environment
- Scoring: No
- AutoTest Cap.: No
- Reposition Cap.: Yes
- Motion Fidelity: 6 DOF
- Engine Type: Turbine
- Air Vehicle Class: Airplane
- Training Sys Name: Flight Instrument

Tacan Self Test Decision Group: Enact
PilotTACSLfTestTerm: No

Accept Decision Help Re-Decide
(bind ?tests_a (create$ TAC TAC VOR VOR))
(bind ?tests_b (create$ S1f Sec S1f Sec))
(bind ?tests_c (create$ SELF SECONDARY SELF SECONDARY))
(bind ?tests_d (create$ TACAN TACAN VOR VOR))

(while (<= ?testtype 4 )
  (bind ?tal (nth ?testtype ?tests_a))
  (bind ?tbl (nth ?testtype ?tests_b))

--;- Check for initiate at: power on, push & release, push &hold
  (bind ?testinit  (sym-cat "Pilot" ?tal ?tbl "TestInit"))
  (bind ?testinitinst (symbol-to-instance-name ?testinit ))
  (bind ?testinitans (send ?testinitinst get-Multi_Entry ))

  (if ( member Power_On ?testinitans ) then
    (bind ?P_INITIATE_AT_POWER_ON TRUE )
  else
    (bind ?P_INITIATE_AT_POWER_ON FALSE )
  )

  (if ( member Push_And_Release ?testinitans ) then
    (bind ?P_INITIATE_PUSH_AND_RELEASE TRUE )
  else
    (bind ?P_INITIATE_PUSH_AND_RELEASE FALSE )
  )

  (if ( member Push_And_Hold ?testinitans ) then
    (bind ?P_INITIATE_PUSH_AND_HOLD TRUE )
  else
    (bind ?P_INITIATE_PUSH_AND_HOLD FALSE )
  )

  (format ?where "%s%ss%n"  ADAPT $P_INITIATE_PUSH_AND_HOLDS $ ?P_INITIATE_PUSH_AND_HOLD )

KAMEL RETRIEVE FUNCTION
1 APPLICATION MODEL/DECISION–VECTOR

1.1 Decision Group: TACAN_SELF_TEST

- PilotTACSlfTestTerm = No
- PilotTACSlfTestInit = Push_And_Hold
- PilotTACSlfTestPhases = 1
- PilotTACCompAffect = VOR
- PilotTACSlfPhase1Ch = Channel_Select_Indicator X/Y_Indicator
- PilotTACSlfPhase1ChSlln = 188.0
- PilotTACSlfPhase1ChXYInd = X
Process Support Development

• Concepts:
  - Translates and "load" Domain Engineering work products in the SEE
  - Procedural standards

• Example Work Products
  - Application Engineering Process Standards
  - Application Engineering User’s Guide
  - Application Engineering Environment
  - Application Engineering Environment Support Manual
  - Application Engineering Training Course

• Success indications
  - Application Engineering process has been enacted in the SEE
  - Lessons learned have been documented
    • SEE enactment process must be mature to be followed properly
Look at TACAN power. Don't execute if power is off.

```c
if Power = On then
    -- Include if auto self test at power up.
    --if SP_INITIATE_AT_POWER_ONS THEN
        if New_Power = base_types.true then
            Determined_Active := true;
        end if;
    --Send IF

    -- Include if type is push and release
    --if SP_INITIATE_PUSH_AND_RELEASES THEN
        if Test_Activate = On then
            -- if "button" is currently pressed.
            Determined_Active := true;

            -- Include if button restarts the test if test is already running.
            --if SP_AFFECT_OF_REPRESS_RESTARTS THEN
                if Last_Pass_Active then
                    -- if self test is active
                    Timer := 0.0;
                    -- reset timer.
                end if;
            --Send IF

            --if SP_AFFECT_OF_REPRESS_TERMINATES THEN
                if Last_Pass_Active then
                    -- if self test is active
                    Determined_Active := false; -- Deactivate Self Test.
                end if;
            --Send IF
        else
            -- if active from last pass let the timer terminate the test.
            if Last_Pass_Active then
                -- if active from last pass let
                Determined_Active := true;
            end if;
        end if;
    --Send IF

    -- Include if type button is push and hold.
    --if SP_INITIATE_PUSH_AND_HOLDS THEN
        if Test_Activate = On then
            Determined_Active := true;
        else
            Determined_Active := false;
        end if;
    --Send IF

    -- Include if change in channel interrupts self test
    --if SP_TERM_WITH_CHANS THEN
        if Change_In_Channel then
            Determined_Active := false;
        end if;
    --Send IF

    if Determined_Active then
        -- If first pass of a self test then initialize timer.
        if not Last_Pass_Active then
            Timer := 0.0;
        end if;

        -- Increment self test timer.
        Timer := Timer - Iteration_Duration;

        -- Search phase end time array to determine what phase self test is in
        for I in 1..Num_Phases loop
```

ADAPTABLE CODE
-- SELF_TEST retrieve file

begin

ADAPT $P_PACKAGE_TEST_NAMES$   DETERMINE_TACAN_SELF_TEST
ADAPT $P_PROCEDURE_TEST_NAME$   PROCESS_TACAN_SELF_TEST
ADAPT $P_TERM_WITH_CHAN$        FALSE
ADAPT $P_EXTERNAL_OUTPUTS$      TRUE
ADAPT $P_NUM_PHASES$             "1"
ADAPT $P_END_TIME_OF_PHASES$     "0.0"
ADAPT $P_INITIATE_AT_POWER_ON$   FALSE
ADAPT $P_INITIATE_PUSH_AND_RELEASE$ FALSE
ADAPT $P_INITIATE_PUSH_AND_HOLD$ TRUE
ADAPT $P_AFFECT_OF_REPRESS_RESTART$ FALSE
ADAPT $P_AFFECT_OF_REPRESS_TERMINATE$ FALSE
ADAPT $P_AFFECT_OF_REPRESS_NO_EFFECT$ FALSE
ADAPT $P_ST_FAILS$   FALSE

COPY type SELF_TEST_BODY_A. TACAN_SELF_Test.adb
COPY type SELF_TEST_SPEC_A. TACAN_SELF_Test.ads

end
Declare variables required within package.

-- Number of phases
Num_Phases : Base_Types.Signed_Integer_16 := 1;
Phase : Base_Types.Signed_Integer_16 := 0;
Iteration_Duration : float_32 := 0.1;

Timer : float_32 := 0.0;
Last_Pass_Active := false;
Test_Active : Base_Types.Discrete_State;

-- Define and initialize array that contains end times for each self test phase.
-- These are elapsed time from self test initiation that each phase will
-- terminate.

type float_array is array (Base_Types.Signed_Integer_16 range <>) of float_32;
Phase_End_Time : float_array (1..1) :=
  0.0;

procedure PROCESS_TACAN_SECONDARY_TEST(
  -- Declare variables for this procedure
  TestActivate : in Base_Types.Discrete_State;
  Power : in Base_Types.Sim_Boolean;
  New_Power : in Sim_Boolean;
  External_Test_Out: out Base_Types.Discrete_State;
  Test_Phase : out Base_Types.Signed_Integer_16)

  -- Declare and initialize local variables.
  Phase_Found : Boolean := false;-- Phase Found flag.
  Determined_Active : Boolean := false;

begin
  -- Initialize some variables.
  Phase_Found := false;
  Determined_Active := false;

  -- Look at TACAN power. Don't execute if power is off.
  if Power = True then
    if Test_Activate = On then
      Determined_Active := true;
    else
      Determined_Active := false;
    end if;
  end if;

  if Determined_Active then
    -- If first pass of a self test then initialize timer.
    if not Last_Pass_Active then
      Timer := 0.0;
    end if;

    -- Increment self test timer.
  end if;
Rockwell International
Command and Control Systems Division

Synthesis Pilot Project

Presented by
Jerri Turner–Harris
DOMAINS / SUBDOMAINS

- COMMUNICATIONS MANAGEMENT AND CONTROL DOMAIN

- MIL-STD-1553B INTERPROCESSOR COMMUNICATION SUBDOMAIN

- MESSAGE HANDLING SYSTEM DOMAIN
LESSONS LEARNED

- ADOPT DISCIPLINED METHODS
  REAL-TIME STRUCTURED ANALYSIS
  ADARTS
  CODING AND DOCUMENTATION STANDARDS
  METRICS

- COMMUNICATE DOMAIN EXPERIENCE
  PRODUCT REVIEWS

- USE AVAILABLE AUTOMATED TOOLS
  OPENSELECT
  TEAMWORK
  WORDPERFECT MACRO AND MERGE
  TRF
The Role of ADARTS in Domain Analysis

System Level

- Domain Analysis
- Real-time Structured Analysis
- Partitioning
- Configuring
- Real-time Structured Analysis

Software (CSCI) Level

- Process Structuring
- Class Structuring
- Information Modeling
- Software Architecture Design
- Ada Architecture Design

Work Products

- Synopsis
- Glossary
- Common Requirements
- Variable Requirements
- Decision Model
- Graphical Decision Model

Activities

- Needs Allocation Diagram
- Configuration Diagram
- Node Allocation Diagram
- Node Behavior Specifications
- Message Interface Specifications
- Hardware Specifications

Source: SPS Adatas Guidebook, v. 5.00, 1993
SYNTLDKW 01/01/94 DJU
Synopsis Segment

Subdomain implementations provide error free application to application communications by MIL-STD-1553B bus. Subdomain implementations may provide 1553B bus control, 1553B remote terminal and 1553B bus monitoring functions.
Commonality

Every MHS will periodically delete expired messages from storage.

Variability

- How long a message can be stored.
- How often will storage be checked for expired messages.
- What time will storage be checked.

Decision Model

<table>
<thead>
<tr>
<th>Description</th>
<th>Decision</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>How long a message can be stored.</td>
<td>selection of (7 days, 30 days, 60 days, 90 days)</td>
<td>[STORAGE_LENGTH]</td>
</tr>
<tr>
<td>How often will storage be checked for expired messages.</td>
<td>selection of (once a day, once a week, once a month)</td>
<td>[STORAGE_CHECK]</td>
</tr>
<tr>
<td>What time will storage be checked.</td>
<td>[(1:00..12:00)] selection of (am, pm)</td>
<td>[STORAGE_TIME] [STORAGE_AM_PM]</td>
</tr>
</tbody>
</table>
Graphical Decision Model Segment

- Operator Console
  - <<CONSOLE>>

- True
- False

- Operator Console
  - Decisions
Terminal Information

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Terminal Identifier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Terminal address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Role</td>
<td>Bus Controller (BC)</td>
<td></td>
</tr>
<tr>
<td>4. Redundancy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Broadcast</td>
<td>Remote Terminal (RT)</td>
<td></td>
</tr>
<tr>
<td>6. Processor</td>
<td>Bus Monitor</td>
<td></td>
</tr>
<tr>
<td>7. Ram</td>
<td>BC/RT</td>
<td>4</td>
</tr>
<tr>
<td>8. Rom</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>9. 1553B chip</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10. Interface</td>
<td></td>
<td>Shared Memory</td>
</tr>
<tr>
<td>11. Diagnostic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12. Bus Assignment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13. Polling Sequence</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>14. Save Terminal</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Enter Selection:
Questions or comments on content should be directed to:

K.C. King
Boeing/STARS Demonstration Program Manager
% Dual Inc.
30 Skyline Drive
Lake Mary, FL 32746
(407) 333-8880

Jerri Turner-Harris
Rockwell Command & Control Systems Division
MS 460-220
3200 East Renner Road
Richardson, TX 75082-2402
(214) 705-3151

Or to:

Grady Campbell
Software Productivity Consortium
2214 Rock Hill Road
Herndon, VA 22070
(703) 742-7104

Send feedback on the Consortium’s Video Program and orders for video products to:

Technology Transfer Clearinghouse
Software Productivity Consortium
2214 Rock Hill Road
Herndon, VA 22070
(703) 742-7211