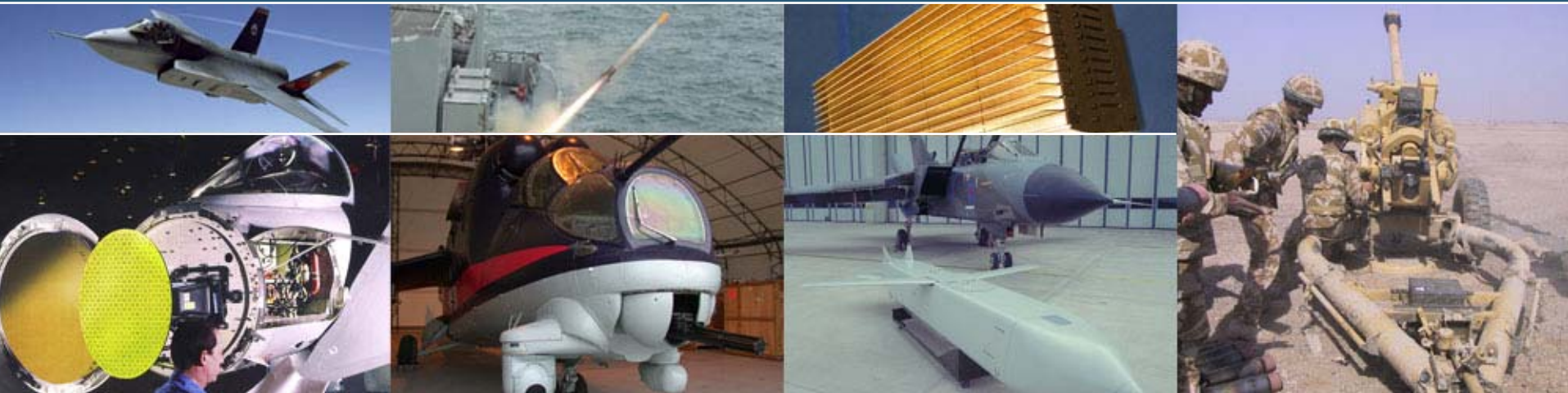


Stateful Data Processing in a Distributed Environment

Douglas Swanson

CAPTOR Tranche 2 Systems & Software IPT
BAE SYSTEMS Avionics, Sensor Systems Division



Overview

Introduction

Characteristics of State

Local Variables

Feedback Loops

Segmentation

External State

Reset Levels

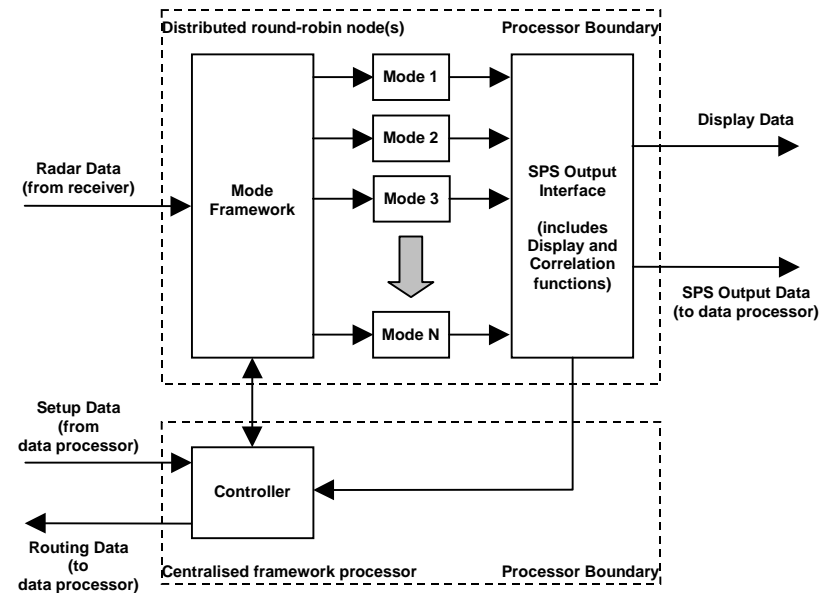
Multiple Primitive Access

Conclusions



Introduction

- The CAPTOR Tranche 2 (T2) programme is developing the multi-mode nose radar for the Eurofighter Typhoon
- The T2 Signal Processing Software (SPS) is designed to operate on a distributed, multi-target hardware platform
- Radar data is processed by modes on round-robin processing engines to maximise performance and throughput
- This mode-based approach is well-suited to token-based data flow used by Gedae
- Must also be able to maintain some data across multiple firings of modes - state data
 - Successive firings of the modes may occur on physically distributed processors with separate memories

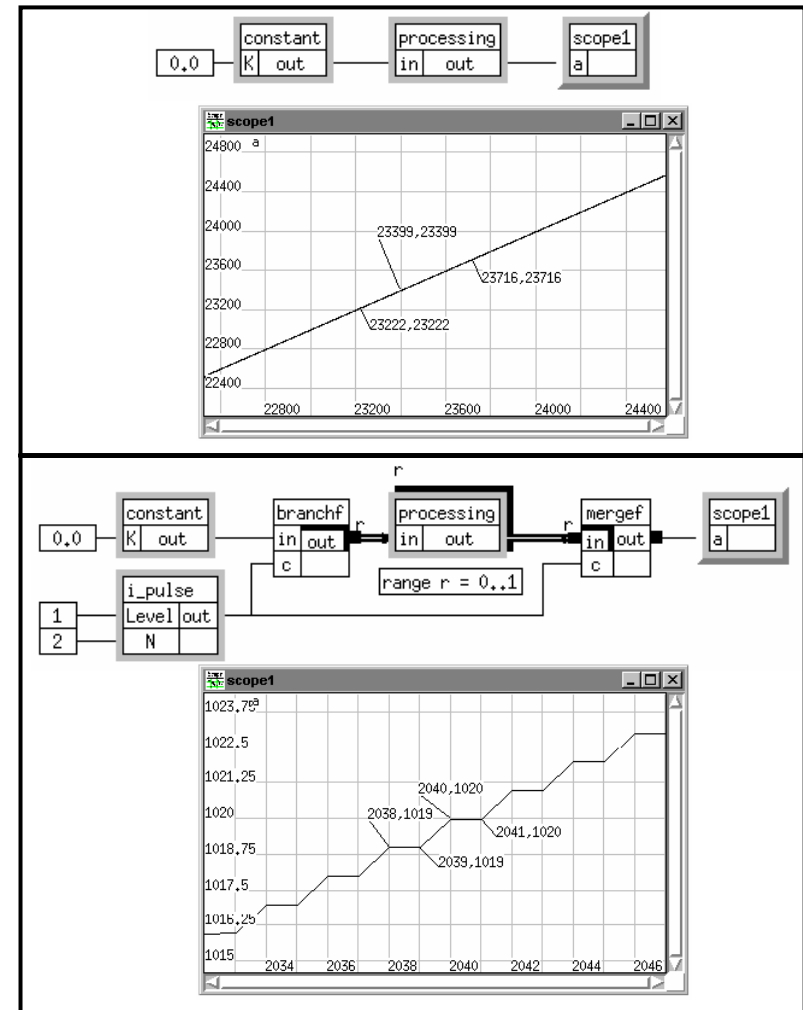


Characteristics of State

- In its simplest form, state data is data with a history
 - Previous value of state data and other current inputs used to calculate next value of state data
- Additional complexity is introduced when multiple instances of state data are required
 - e.g. when functionality is replicated in multiple round-robin nodes distributed across multiple processors
 - Efficient implementation required that maximises achievable parallelism of round-robin nodes
- Further complexity is added when the state data must be accessed multiple times in an instance
 - e.g. state data is used by multiple primitives in a flowgraph
 - Order of access is crucial to correct behaviour
 - This is typical of the use of state data by the algorithms comprising the radar modes
- Different types of state are required in different places within the SPS

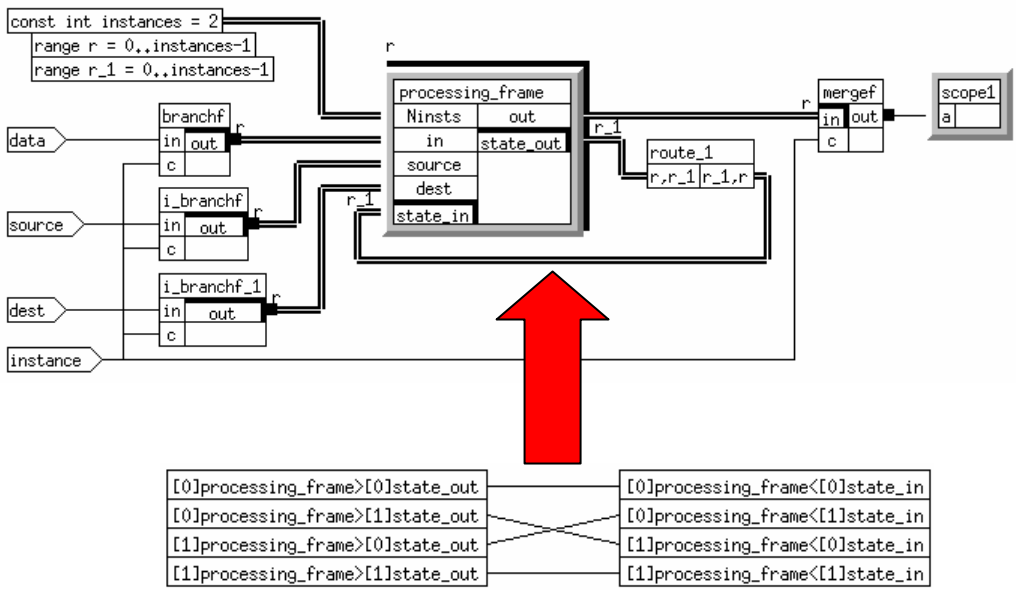
Local Variables

- Local variables can be used to implement state for a single primitive
 - Data is persistent across multiple firings of the box
 - Can be initialised by Reset method
- Cannot be used in distributed application
 - Each instance of the box has its own copy of the data which is only updated when that instance of the box fires
- In the T2 SPS, modes are farmed out to round-robin processing nodes, each of which has its own copy of all of the modes and all of the state data
 - Unless successive modes are run on the same processor, state data will not be correctly updated
 - Local variables are not suitable for this



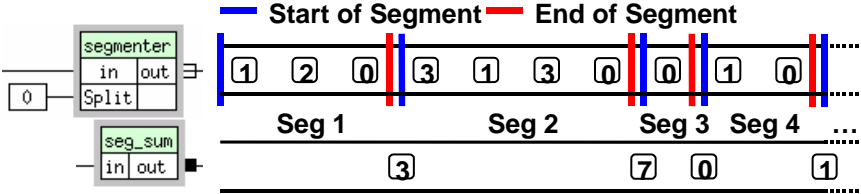
Feedback Loops

- To correctly handle multiple uses of state data within the application there are two choices:
 - Route state data through a centralised processor
 - Easier to manage data
 - Potential processing bottleneck
 - Route state data directly to the next processor that will need it
 - Must tell processors where to get state data from, where to send it to and when to reset it
 - Requires N^2 connections



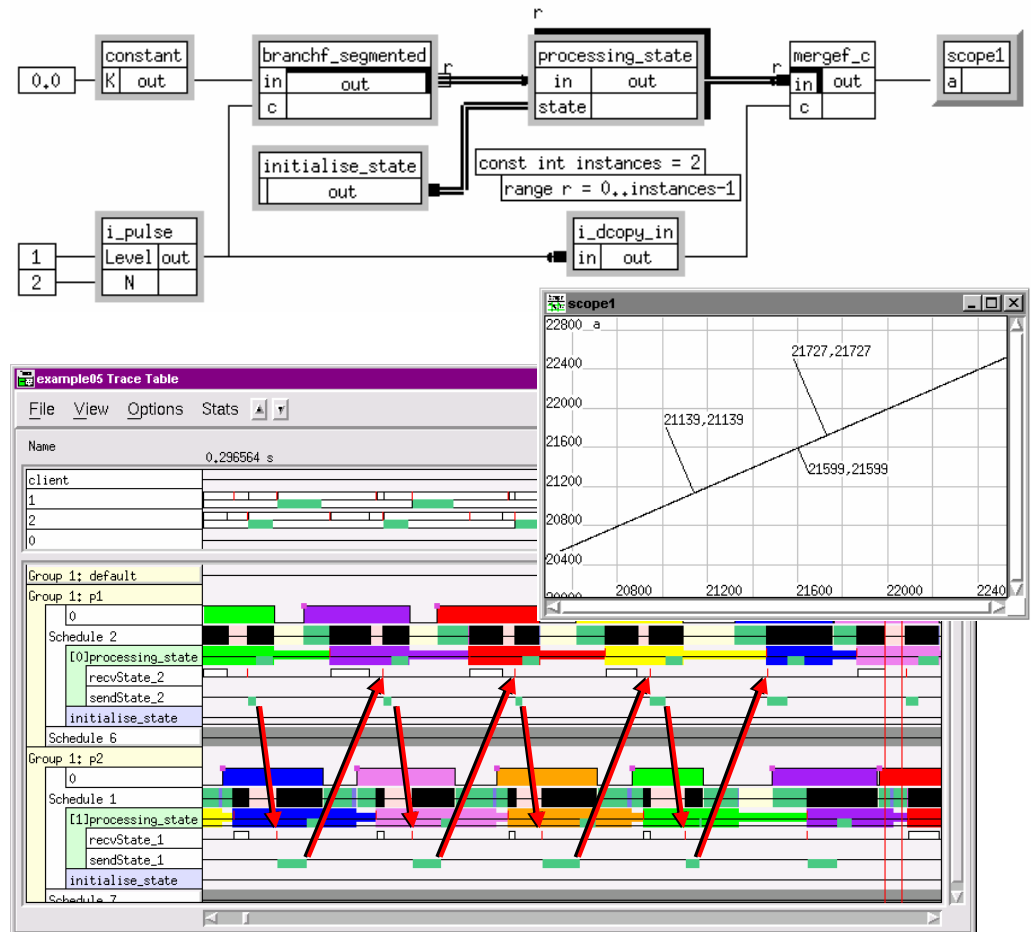
Segmentation

- Enabling capability for other advanced features: external state, exclusion
- Permits runtime delimitation of subsets of stream data
 - Implicit ordering defined at segment boundaries
 - Can run specific processing at start or end of segment, e.g. state reset
- Scope extends only to next box (flowgraph or primitive) following segmenter
- Processing of each exclusive segment on any one processor must be complete before next can commence
 - Can identify corresponding data in all other inputs and outputs
 - Provides flow control and current state of application
 - To avoid pipelining of segmented application, state must be treated differently from streams



External State

- State primitive boxes:
 - Initial value set by Reset Method at start of each segment
 - Must be carefully positioned at appropriate level in segmented hierarchy to give correct reset behaviour
- Strict “order of access” control enforced, based on segment number
- sendState and recvState boxes and connections automatically inserted and managed by Gedae
- Used extensively in T2 SPS
 - Efficient, elegant and understandable design



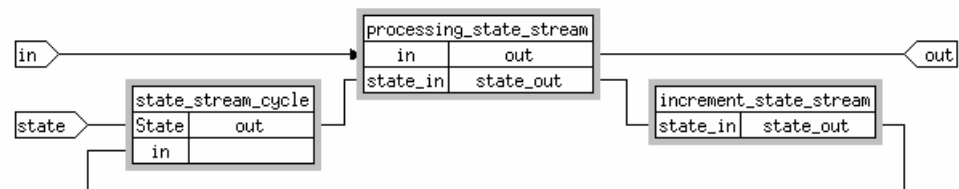
Reset Levels

- Careful positioning of state boxes within segmented hierarchy is crucial to ensure correct reset behaviour of state data
 - Hierarchy must be considered in advance and designed into application
 - Otherwise, can incur significant rework
- T2 SPS uses many levels of exclusive segmentation
 - Radar mode groups
 - Radar modes
 - Instances of radar modes
 - Enumerations of instances
- State data can be shared by enumerated instances of a mode, at any level desired
 - So can schedule memory



Multiple Primitive Access

- Order of access to state data for reading and writing is also crucial to correct behaviour of application
 - Care required when one or more boxes within the scope of a segment must access state data
 - Within that scope, can convert the state data into a stream to ensure correct order of processing by multiple boxes, and feed back tokens generated to update state
 - Cyclic primitives can be used to enforce the update and use of the state data in strict rotation
 - Propagation of state update to other instances handled automatically by Gedae



```

Embeddable Box Editor
File Bookmarks Edit Utilities Help
Class Name: user/state_stream_cycle

Name: state_stream_cycle
Type: cyclic
Comment: "Cyclically copy State input to stream output,
and then stream input to State"
-
Input: {
  stream int in(Nin);
  state int State;
}
Local: {
  int Nin;
  int Nout;
  int even;
}
Output: {
  stream int out(Nout);
}
Length: {
  return 2;
}
Cycle: {
  Nin = firing/2;
  Nout = (firing+1)/2;
}
Start: {
  even = 1;
}
Apply: {
  int i, G = granularity;
  for (i = 0; i < G; i++) {
    if (even) {
      /* Copy from State to out */
      memcpy(&(out[i/2]), &(State), sizeof(int));
      even = 0;
    } else {
      /* Copy from in to State */
      memcpy(&(State), &(in[i/2]), sizeof(int));
      even = 1;
    }
  }
}
Status: loaded file Text: unchanged
  
```

Conclusions

- Several methods exist by which state data can be implemented in Gedae
 - Each has its own merits, but some have limited applicability
 - Local variables: easy to implement, cannot be distributed
 - Feedback loops: support distribution and multiple primitive access, untidy graphs and high maintenance effort
 - External state: support distribution and memory sharing, can use cyclic primitives to allow multiple primitive access, tidy graphs with lower maintenance effort, require well-structured graph hierarchy
- State used extensively in CAPTOR Tranche 2 Signal Processing Software
 - Local variables for non-distributed state (framework controller)
 - External state for distributed round-robin mode processing
- Hierarchical graph structure offers other related benefits:
 - Exclusion (resource saving memory sharing)
 - Powerful, flexible processing due to regular, scaleable structure of application
 - Clear, efficient and easily maintainable design

- Any Questions?