STITCHES
SoS Technology Integration Tool Chain for Heterogeneous Electronic Systems

Dr. Evan Fortunato

This Research was developed with funding from the Defense Advanced Research Projects Agency (DARPA)

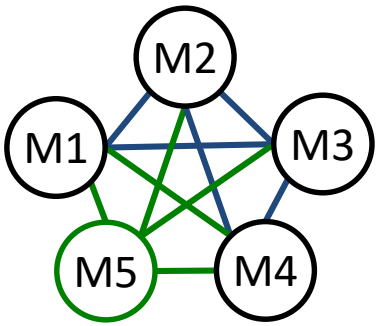
The Goal: Composing Systems That Keep Up With The Times

APOGEE RESEARCH

- DoD has long assumed that homogeneous, fixed-configuration weapon systems are the only way to meet their goals of a superior military force
  - Must last a long time, so requirements are developed for 30+ years out.
  - Meeting 30 year out requirements with today's technology is hard
  - **Result is the best design possible with 20-30 year old technology** and updates are not efficient with respect to time or cost…
- Open Architectures Try to Solve this Problem
  - Requires enormous effort to reach a "global" consensus on the system architecture,
    - Even then, it is only a "local" version of "global"
    - Global standards have to work for everyone, so aren't optimized for your application
  - Result is **heterogeneous components in a homogeneous architecture** – which doesn't work because the architecture needs to evolve with the technology
  - Attempts to build flexibility into the architecture (to support heterogeneity) just result in overly complex infrastructures that still don't anticipate the new technologies
- What if Global Interoperability Didn't Require a Common Interface at ALL?

Understanding the Trade between Local and Global Message Standards…

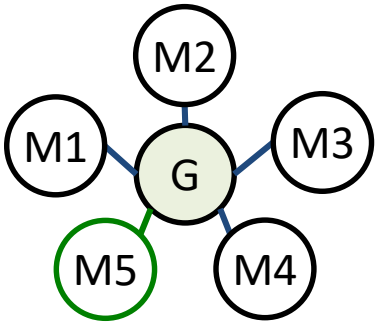
- Local Message Standards
  - Flexible – You Can Add New Messages Easily
  - Inefficient - Require N^2 Transforms (all pairs) for Interoperability



M# = Message #

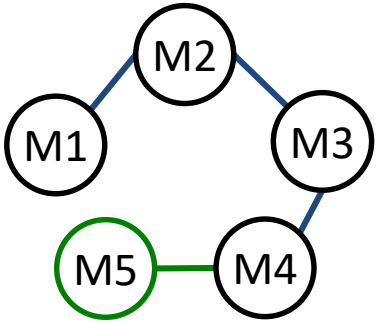
Transform M2 <- M1:
M2 = T21(M1)
Transform M5 <- M1:
M5 = T51(M1)

- Global Open Standards
  - Efficient – N Transforms to/from the Global Standard
  - Not Flexible – Can't change without tremendous effort



Transform M2 <- M1:
M2 = T2G(TG1(M1))
Transform M5 <- M1:
M5 = T5G(TG1(M1))

- Incremental Standards (STITCHES)
  - Efficient – ~N Transforms for Interoperability
  - Flexible – You Can Add New Messages Easily



Transform M2 <- M1:
M2 = T21(M1))
Transform M5 <- M1:
M5 = T54(T43(T32(T21(M1))))

Key Innovation: Field and Transform Graph (FTG)

- Fields are Nodes in the Graph and Contain:
  - A set of subfields (which are defined by other nodes in the graph)
  - A set of properties (mathematically precise specification of node properties)
  - Note: All node information is defined locally, no coordination required!
- Nodes are Connected by Links That Define the Transform from Source to Destination Nodes
  - Each link requires a pair wise human coordination between the source and destination
  - Transforms Expressed in a Domain Specific Language Built for this Purpose
  - Graph algorithms determine a composition of transforms (path through the FTG) that produce the destination message given a source message
- No Global Coordination Required to Update or Evolve Data in the FTG

STITCHES Uses a Domain Specific Language to Capture the Specification

- Why a Domain Specific Language (DSL)?
  - General Purpose Languages are Hard to Read, Write and Formally Analyze
  - Standard System Engineering Tools Are Limited
  - DSL Provides a Narrow Language That is Tailored To This Problem
- Assign is the Key STITCHES Capability to Allow Efficient Use of the FTG
  - Assign designates that two instances represent the same thing in the real world
  - STITCHES then finds a path through the FTG to convert between the fields
  - Result is that even if the messages don't match, many of their subfields will

Sensor Output Message: Sensor.Out

```
Time:Sensor.Time
Dets:Sensor.Detection[ ]
```

XForm

Tracker Input Message: Tracker.In

```
DwellTime:Tracker.Time
Contacts: Tracker.Detection[ ]
```

```
XForm(in:Sensor.Out):Tracker.In {
  DwellTime = Assign(In.Time);
  Contacts = Assign(in.Dets);
}
```

Resolve

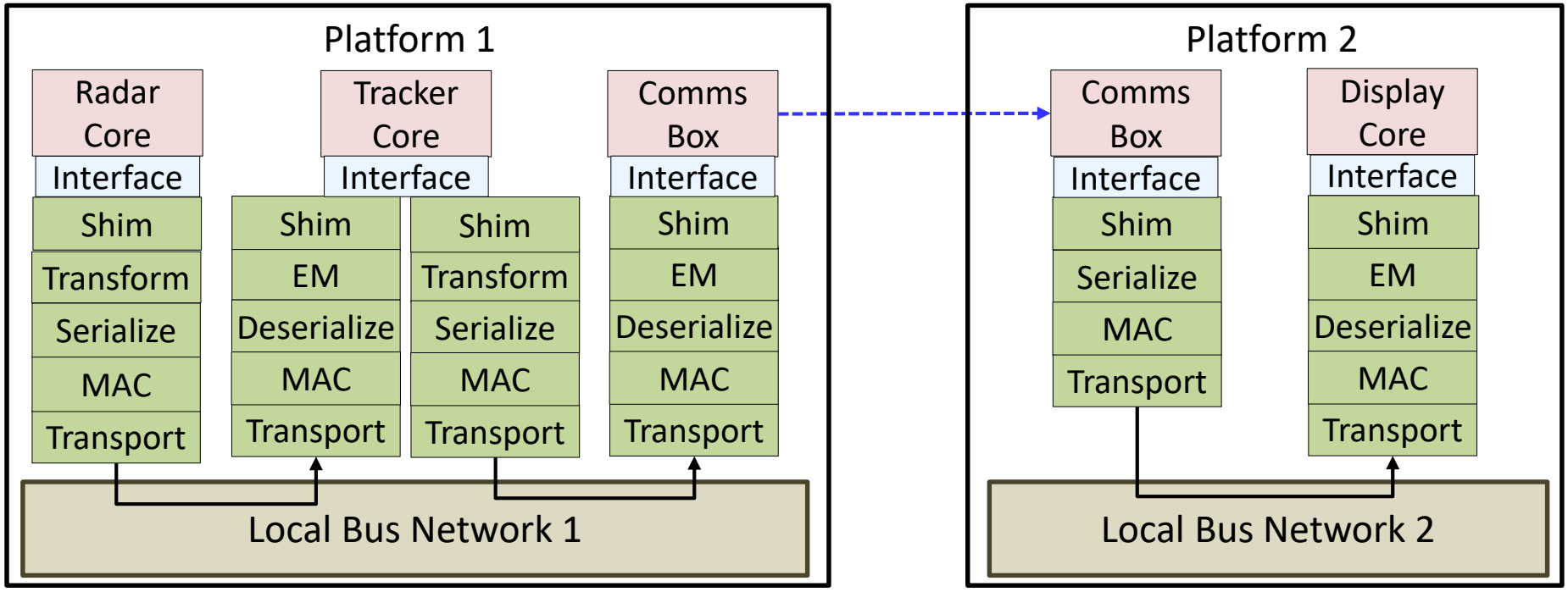
```
XForm(in:Sensor.Out):Tracker.In {
  DwellTime = In.Time-18;
  Contacts = in.Dets*180/3.14159;
}
```

What Does STITCHES Produce?

Subsystem Core — Developed by Subsystem Engineers

Interface — Subsystem Interface, Developed by SS Engineer with STITCHES Autogenerated Libraries. Developed once per Core Version to Work for all SoS Configurations

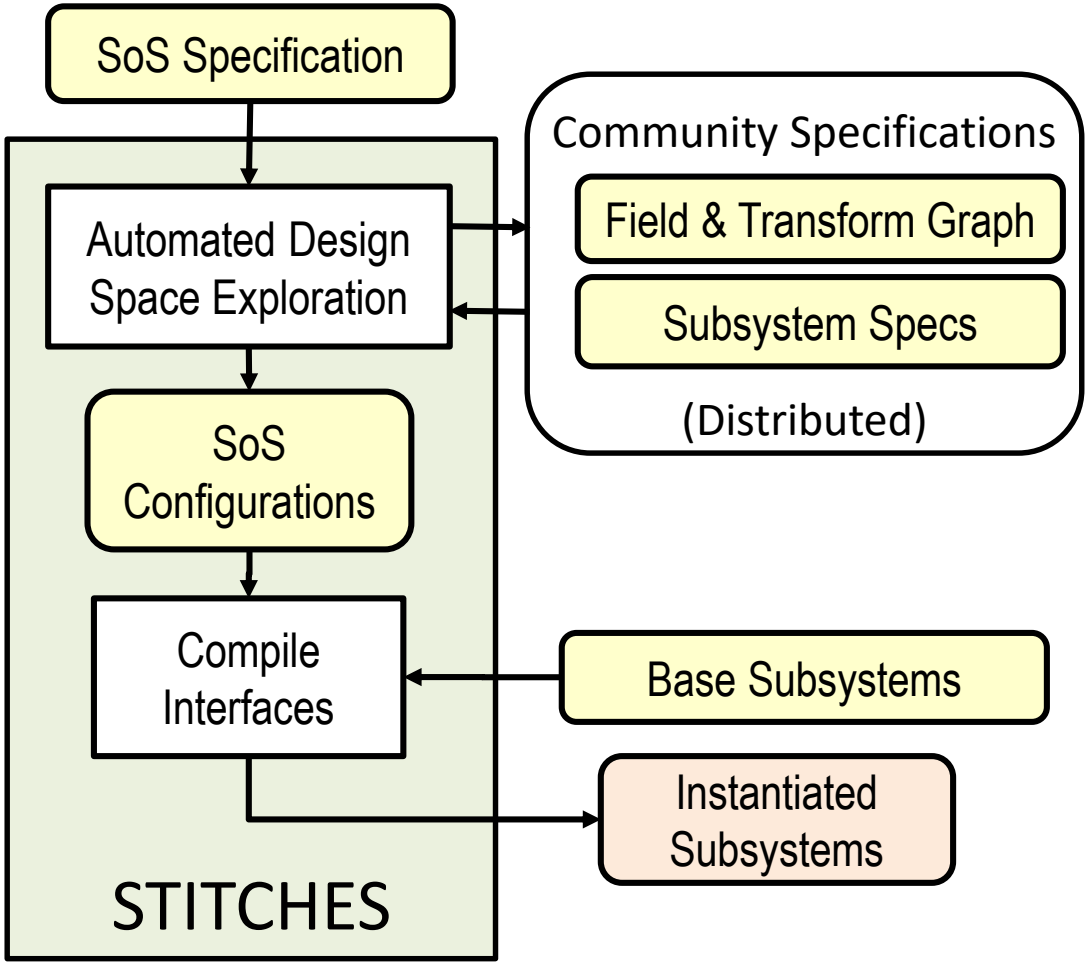
Generated Glue Code — Autogenerated by STITCHES; Tailored to Each SS and SoS Configuration



MAC: Message Authentication Code
EM: Execution Monitor

STITCHES is Focused on Implementing a Scalable Approach to Building SoS Capabilities

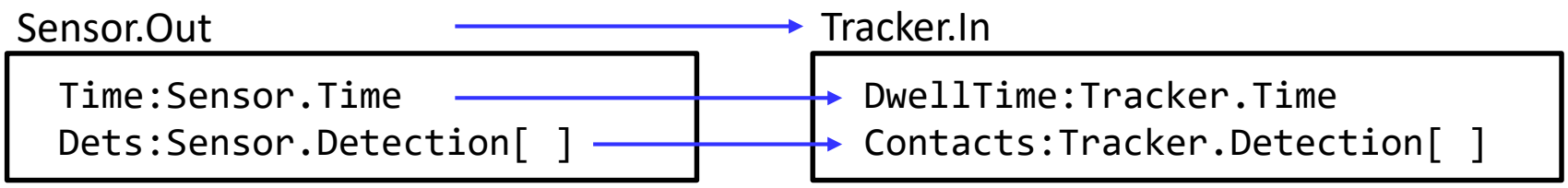
- Design Space Exploration
  - Process FTG to Construct Transformation Chains
  - Specify Interface Stack by forming & solving optimization problems
- Compiler
  - Construct Interface Stack Structure
  - Optimize Transforms for this Instance of the Interface
  - Provide Cyber Security through whitelist property enforcement
  - Generate C++/Java Code & Compile into binaries



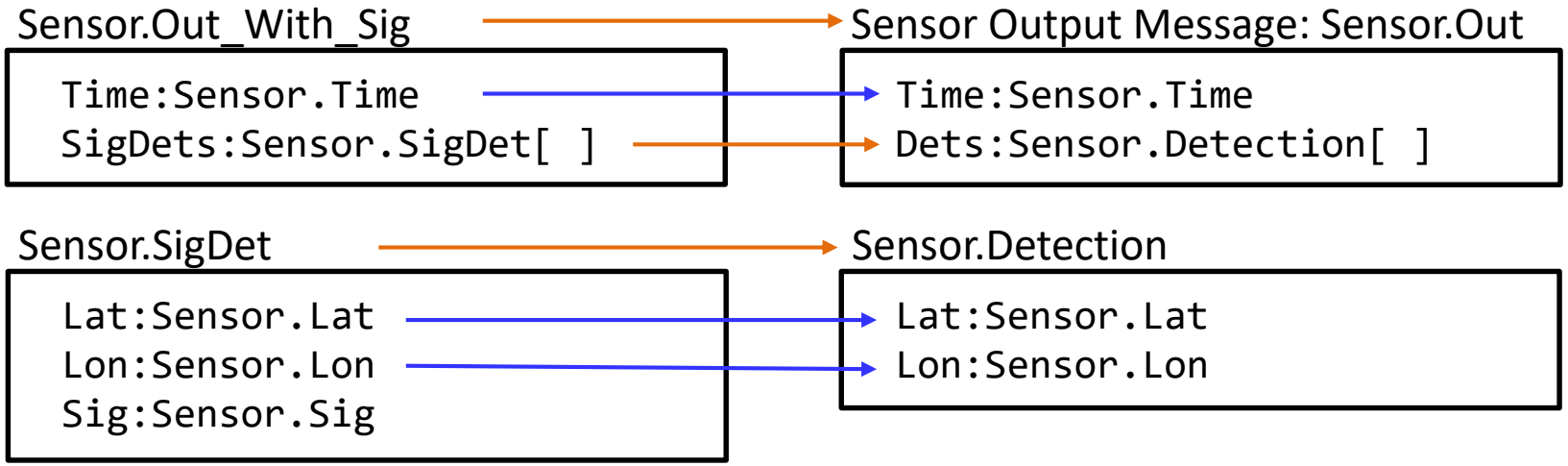
Result: High Performance Interfaces Optimized For Each Application

Evolution of the Architecture: Backwards Compatibility

We Started with a Sensor_Out and a Tracker_In Message



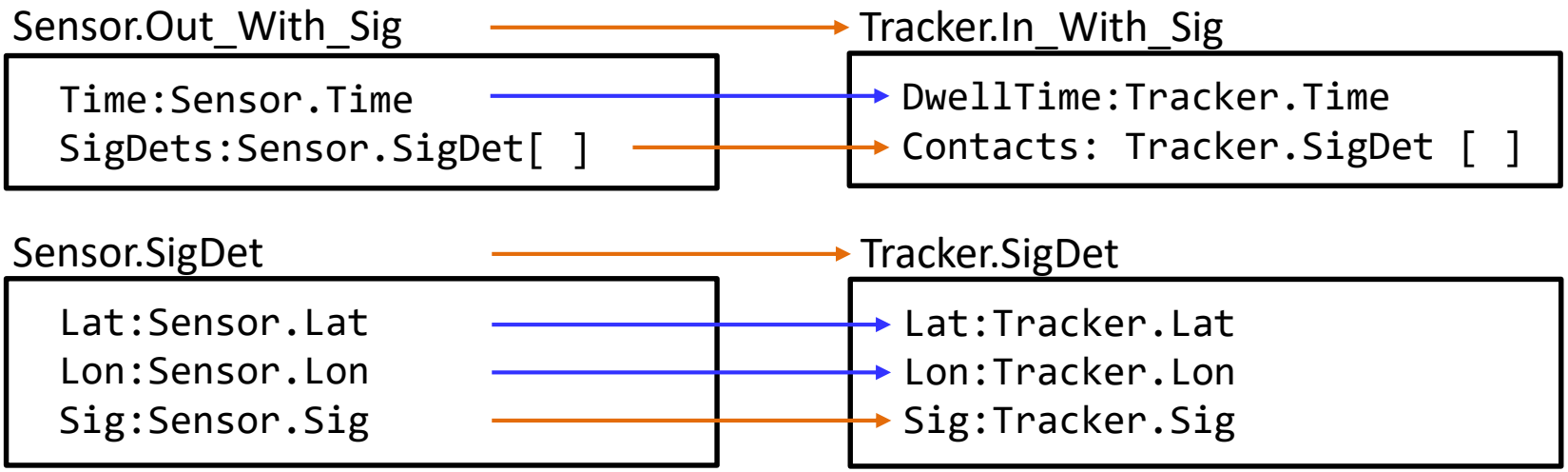
Next Add a Sensor_Out_With_Sig and connect it to the Sensor_Out



New Sensor with Signatures Can Now Interoperate with the Tracker
But Tracker Doesn't Use the Signature Information

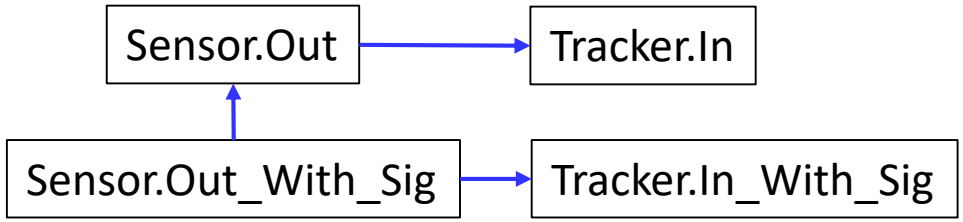
Evolution of the Architecture: Forwards Compatibility

- Now Let's Add in an Upgraded Tracker (that can use the Signatures)



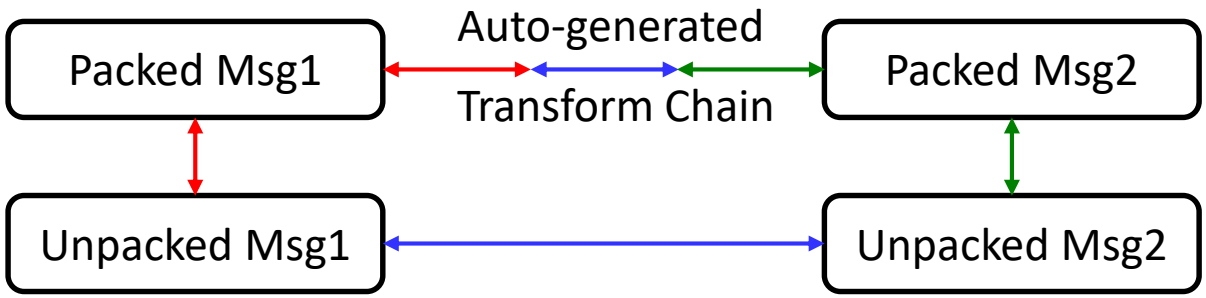
New Tracker Can Now Use the Full Data from the New Sensor (Including Signatures)

Updated FTG Supports the Old, New and Mixed Architectures



Handling Packed Representations

- Many real systems mix their interface definition with their implementation
  - Result is a serialized (Packed) form of the interface that can represent multiple different interface messages (e.g., STANAG 4607) with descriptor words for run time resolution
  - Packed messages are often used for run-time efficiency - they tend to be the big / high rate messages in the system. So don't want to unpack if not necessary
- Mirrored Unpacked Nodes Provide an Effective and Efficiency Solution
  - Create a Second Unpacked Node that Contains a Structured Version of the Interface
  - Create Transforms between the Unpacked and Packed Nodes
  - Interact with other Interfaces via their Unpacked Representations
  - Auto-generate the Desired (high performance) Packed-to-Packed Transforms



Optimized Performance:

[Packed → Unpacked → Unpacked → Packed] vs. [Packed → Packed]

APOGEE RESEARCH

Connection	PUUP vs PP	Java Implementation		C++ Implementation	
		Speed Mbps	Latency ms	Speed Mbps	Latency ms
R1 -> T1 (No Transform)	PUUP	2956	1.1	2818	0.7
R1 -> T1 (No Transform)	PP	2956	1.1	2818	0.7
R1 -> T2 (Only Change Time)	PUUP	2783	1.2	2803	0.8
R1 -> T2 (Only Change Time)	PP	2783	1.2	2743	0.8
R1 -> T3 (Switch Order Lat, Lon)	PUUP	1789	1.4	1773	0.9
R1 -> T3 (Switch Order Lat, Lon)	PP	1804	1.4	1747	0.9
R1 -> T4 (Change All Fields)	PUUP	1245	1.6	1555	0.9
R1 -> T4 (Change All Fields)	PP	1228	1.6	1643	1.0

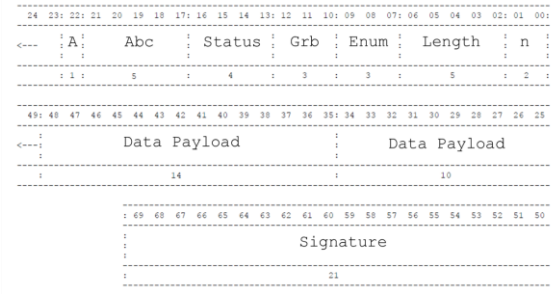
MAC and Execution Monitors are Disabled for these Performance Runs
All interactions via localhost, so no network latencies are involved
Data Gathered on a Standard (~4 Year Old) Quad Core Workstation

Interoperability via Legacy Comms, What if desired messages aren't supported?

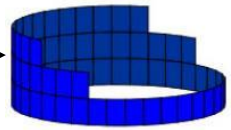
Pod Tracks

?

Legacy Comm Message



Legacy Comm Message

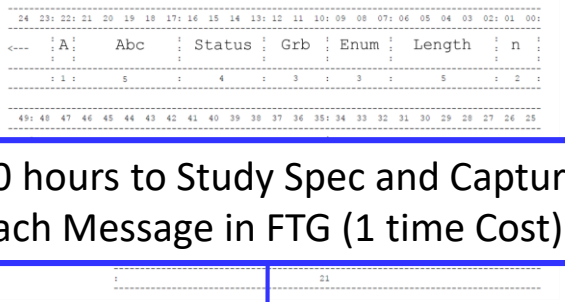


Display Tracks

?

Legacy Comm Message

Auto-generate Transcoder to "Encode" Messages into Legacy Comms Messages



~20 hours to Study Spec and Capture Each Message in FTG (1 time Cost)

Pod Tracks

Serialize

Transcode

Legacy Comm Messages

Legacy Msg (600 Bits)

Transcoder Generator

2 Minutes on Laptop

Transcoder

Free Bits (503)

Fixed Bits (97)

Display Tracks

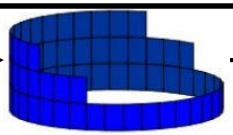
Transform

Pod Tracks

Deserialize

Transcode

Legacy Comm Messages



Manually Defined

Automatically Created

Capability Demonstrated in Live Flight in January 2018 with Example Legacy Comm's System

STITCHES Allows a New Approach to System Integration

- FTG Efficiently Captures Information Required for Interoperability
  - No Global Coordination, Just local and Pairwise Interactions
  - Ontology defined via transformation, not semantics
- STITCHES Allows for the Rapid Instantiation of SoS Capabilities
  - Compile new SoS integrations in minutes, not months
  - Each instance is tailored (optimized) for the needs of that SoS
- STITCHES Is Available to Anyone in DoD As Open Source Toolchain
  - Register for account (with a DoD Gov/contractor email) at www.stitches.tech
  - Full source, install packages, training packages, pre-built VM, user forums, etc.
  - Targeting Public, Open Source Release of STITCHES (not FTG which contains DoD Data)
- STITCHES Is Very Mature for a DARPA Project
  - Dozen Major Releases with large user base.
  - Training sessions since 2015 (300+ people)
  - Used at 8+ major events, including multiple flight events