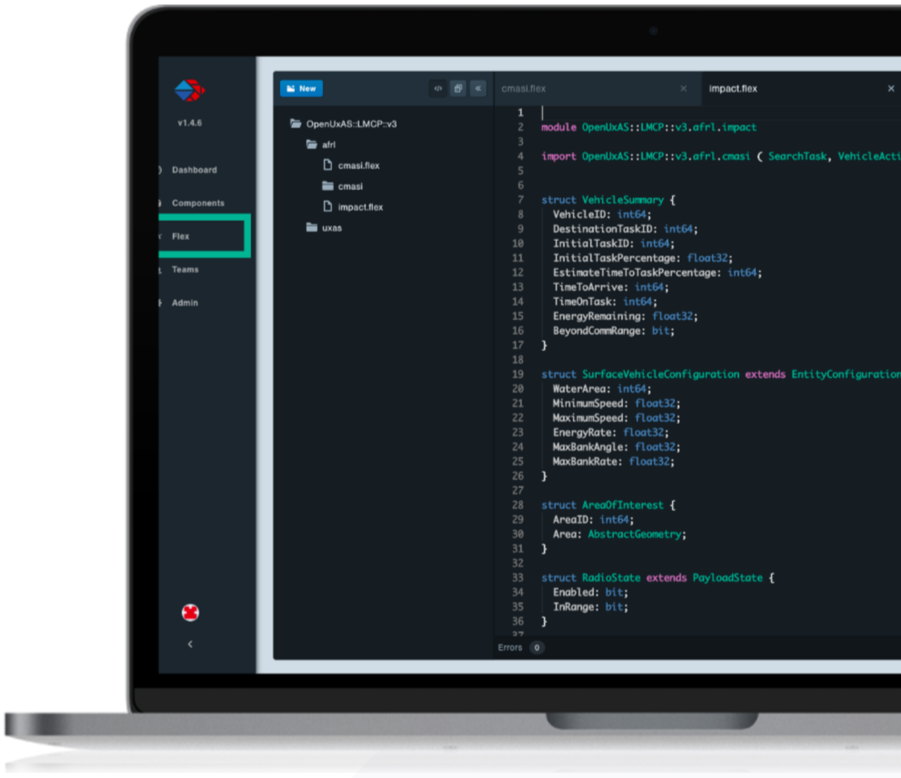
Why Flex?

Technical White Paper
March 2022



```
module OpenUxAS::LMCP::v3.afrl.impact

import OpenUxAS::LMCP::v3.afrl.cmasi ( SearchTask, VehicleActi


struct VehicleSummary {
  VehicleID: int64;
  DestinationTaskID: int64;
  InitialTaskID: int64;
  InitialTaskPercentage: float32;
  EstimateTimeToTaskPercentage: int64;
  TimeToArrive: int64;
  TimeOnTask: int64;
  EnergyRemaining: float32;
  BeyondCommRange: bit;
}

struct SurfaceVehicleConfiguration extends EntityConfiguration
  WaterArea: int64;
  MinimumSpeed: float32;
  MaximumSpeed: float32;
  EnergyRate: float32;
  MaxBankAngle: float32;
  MaxBankRate: float32;
}

struct AreaOfInterest {
  AreaID: int64;
  Area: AbstractGeometry;
}

struct RadioState extends PayloadState {
  Enabled: bit;
  InRange: bit;
}
```

www.tangramflex.com

TANGRAMFLEX®

1. Introduction

1a. Tangram Flex Overview

Tangram Flex, Inc. (Tangram) is a product-driven software company with deep expertise in system modernization, integration, assurance, and autonomy. We believe every mission deserves access to innovation, so we deliver software research, services, and products that enable rapid integration with confidence.

Tangram built and licenses Tangram Pro™, a Component Software Integration Platform (CSIP) that bridges the treacherous gap between system engineering and software development with a practical engineering workspace for assured component deployment.

Tangram Pro is a component-based software engineering (CBSE) tool. Tangram Pro connects software code with system models to dramatically shorten the gaps between phases of the engineering lifecycle. Its many features include the automatic generation of software interface code while generating message transforms that enable otherwise incompatible components to be integrated together, without making changes to the components themselves.

The Tangram Pro platform and other Tangram Flex technologies and services are actively employed in a variety of mission critical settings.

CSIP /sē sip/: Component Software integration platform; a platform toolkit that connects and unifies tools and processes across the Engineering V. CSIPs provide a shared interface for software integration that narrows the gaps between system design and implementation by bringing together requirements, system models, and software code.

1b. Technical Foundations

Rapid Integration with Confidence is Existential for Today's Mission Critical Systems

Many system failures and cyber breaches can be traced to subtle but serious problems in the communication interfaces between different components of an overall system architecture. The DoD environment is especially challenging with an ever-growing list of systems and subsystems coming from different developers, with multiple intellectual property protections, and leveraging different standards, all in use at the same time.

Over the last 10 years, the Defense Advanced Projects Agency (DARPA) recognized the need to pioneer fundamentally new techniques to enable DoD complex systems (and systems of systems) to be both trustworthy and adaptable while meeting the ever-evolving challenges of cyber-security. DARPA initiated two high-risk research programs:

1. **High-Assurance Cyber Military Systems (HACMS)**
   HACMS prototyped mathematical verification techniques and code analysis tools to restructure and cyber-harden DoD systems so they can continue to operate in the face of the strongest possible cyber threats.

2. **Systems of Systems Integration Technology and Experimentation (SoSITE)**
   SoSITE prototyped the STITCHES interface-translation technology as a radical new way to rapidly integrate diverse systems using bespoke legacy interfaces, or disparate modern interface standards.

Together, HACMS and SoSITE blazed trails to show how confidence and rapid integration can be achieved. Tangram Flex was founded by leading DARPA scientists and engineers with the vision of bringing HACMS-like trustworthiness to STITCHES-like rapid integration capabilities. Tangram Flex developed the Flex integration-specification language as an enabler to realize this combined vision.

Learn more about HACMS and SoSITE

- https://www.darpa.mil/program/high-assurance-cyber-military-systems
- https://www.youtube.com/watch?v=OyqNpn6JpBk
- https://www.darpa.mil/program/system-of-systems-integration-technology-and-experimentation

2. Flex: Designed to achieve both rapid integration and confidence

2a. Introduction to Flex

What is Flex?

Flex is a specification language for connecting together multiple software components and/or the software interfaces of hardware components. The Flex language definition is freely available with no restrictions on its use.

Flex enables precise definitions of software interfaces, including both structure and logical properties (such as range restrictions). And because different components may communicate using different formats, protocols, or standards, Flex also enables message translations to be defined between otherwise incompatible software interfaces.

Flex is an integral part of the Tangram Pro CSIP (Component Software Integration Platform). Tangram Pro provides tools for automatically generating software from Flex specifications, both for communication between system components and for translating messages between different formats.

How Does Flex enable rapid integration with confidence?

More and more frequently we need to integrate components in new settings. Flex was created with the goal of integrating components quickly and safely by automatically generating software interfaces that include transforms between interfaces of different formats or standards. Flex provides the following benefits:

- Flex greatly improves the system software engineer's experience of developing integration code between distinct software components
- Flex is ideally suitable for Modular Open Systems Approaches (MOSA)
- Flex tools have been developed for modern DevSecOps and agile software development processes
- Flex-generated component interfaces provide greater confidence and trust in their correctness compared with hand-written code
- Flex has been designed to fit well with rigorous testing and validation protocols

“Every system change introduces brand new, unique problems that no one has ever solved before. Flex removes repetitive, error-prone tasks and frees up energy for human creativity and ingenuity."

- Ben Kyrlach, engineer at Tangram Flex

3. Flex Language in Practice

3a. Flex is Built for Engineers

Flex greatly improves the experience of doing software integration

To integrate software into mission critical systems, engineers need to connect the interfaces of one component to those of another – a process which usually requires a lot of detailed and tricky code to be written by hand. Flex is designed specifically to improve the experience of today's engineers who are designing and developing component software interfaces and transforms to integrate components together.

Flex is easy to read, create, understand, and reason about

Many mission critical systems today are cyber physical systems, typically authored in C++ and Java. While Flex can automatically generate code in many different languages, it was designed with the C++ or Java engineer in mind. Anyone with a C++ or Java background can quickly pick up writing Flex code.

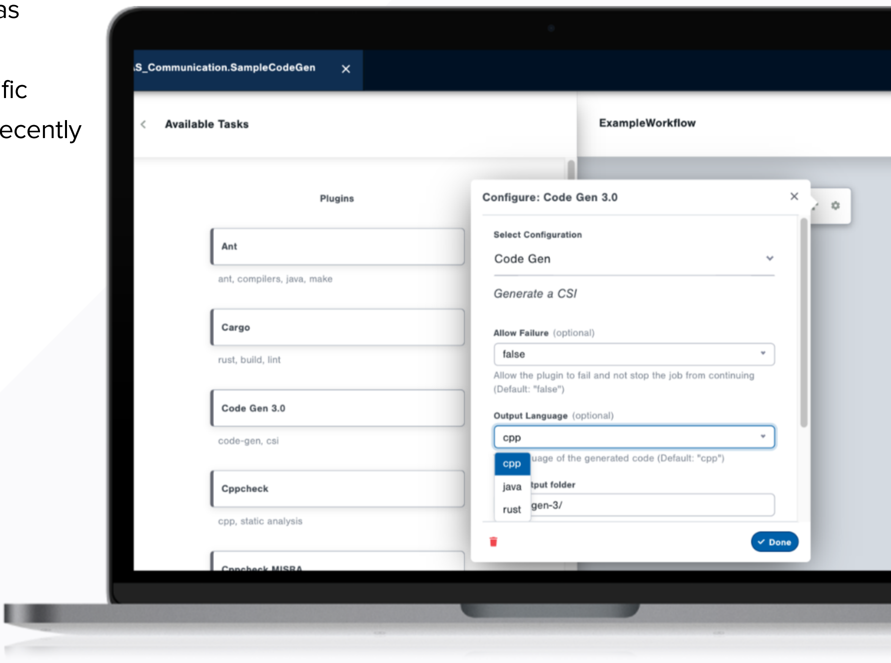
```
struct LocationLLA {
    latitude : float64;
    longitude : float64;
    altitude : float64;
}

message struct WhereAmINow {
    location: LocationLLA;
}

message struct WhereShouldIGo {
    waypoint: LocationLLA;
}
```

Tangram Pro readily and automatically translates Flex code into your language of choice

Tangram Pro leverages the Flex Transpiler in its code generation functionality. A "transpiler" is a tool that generates source code from specifications (or from source code in another language). The Flex Transpiler takes Flex specifications as its input and generates source code in C++, Java, or Rust (additional source code languages can be added as required). C++ was selected to align to cyber physical system C++ requirements. Java was added to meet some specific implementation requirements, and Rust was most recently added to support engineers who are considering migrating from C++ to Rust.



Configure: Code Gen 3.0

Select Configuration: Code Gen

Generate a CSI

Allow Failure (optional): false

Allow the plugin to fail and not stop the job from continuing (Default: "false")

Output Language (optional): cpp

cpp / java / rust

uage of the generated code (Default: "cpp")

tput folder: gen-3/

3b. Flex for Modular Open Systems Approaches

Flex is ideally suited for Modular Open Systems Approaches (MOSA)

MOSA programs require engineers to specify and publish component interfaces. Flex is designed to support the maintenance and publishing of precise, cross-language, interface specifications without referencing the internal IP (intellectual property) of the components. The Flex language itself is non-proprietary, and all Flex specifications belong to their authors and can be licensed with whatever distribution criteria are desired.

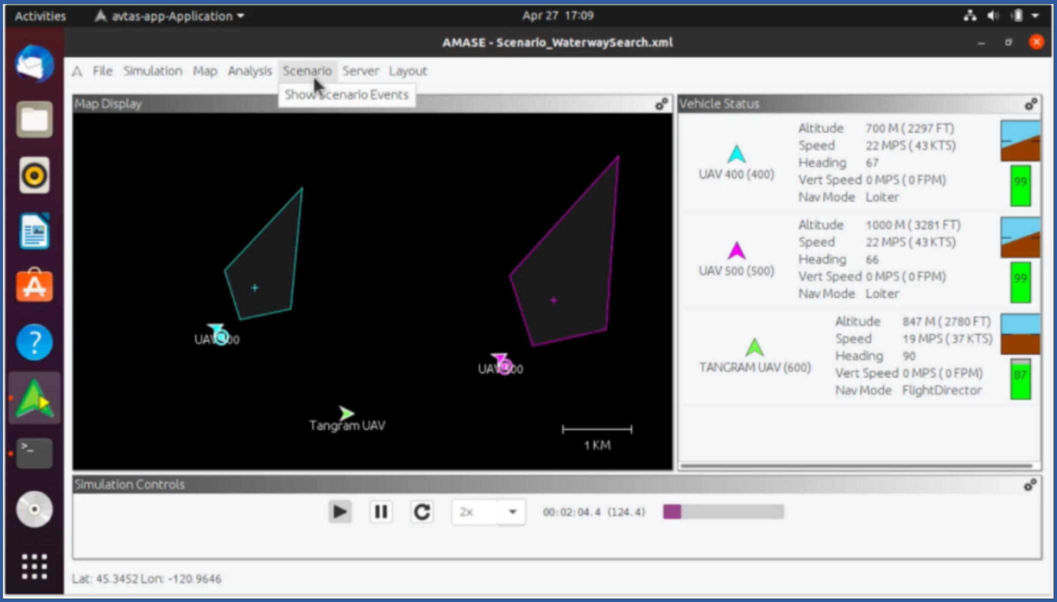
Interfacing to standards is essential to MOSA

Many MOSA programs require conformance to a specific interface standard. Flex is designed to be able to generate interfaces that conform to a wide variety of standards such as OMS or LMCP. Other programs simply require end-to-end connection between interfaces. With Flex, you can do both.

MOSA systems are built using many different transport layers

To allow for greatest modularity and flexibility, a MOSA component should be able to be flexible with regard to how its messages are serialized, or what type of transport is employed, or what type of network configuration is required.

When Tangram Pro auto-generates component software interfaces and transforms from Flex specifications, it allows for just that. It can provide standard interfaces to many different open system standards (or be customized as a one-off, if that's what is needed), using whichever mix of serialization, transport, or networking options are required.



UAV 400 (400)
Altitude 700 M (2297 FT)
Speed 22 MPS (43 KTS)
Heading 67
Vert Speed 0 MPS (0 FPM)
Nav Mode Loiter

UAV 500 (500)
Altitude 1000 M (3281 FT)
Speed 22 MPS (43 KTS)
Heading 66
Vert Speed 0 MPS (0 FPM)
Nav Mode Loiter

TANGRAM UAV (600)
Altitude 847 M (2780 FT)
Speed 19 MPS (37 KTS)
Heading 90
Vert Speed 0 MPS (0 FPM)
Nav Mode FlightDirector

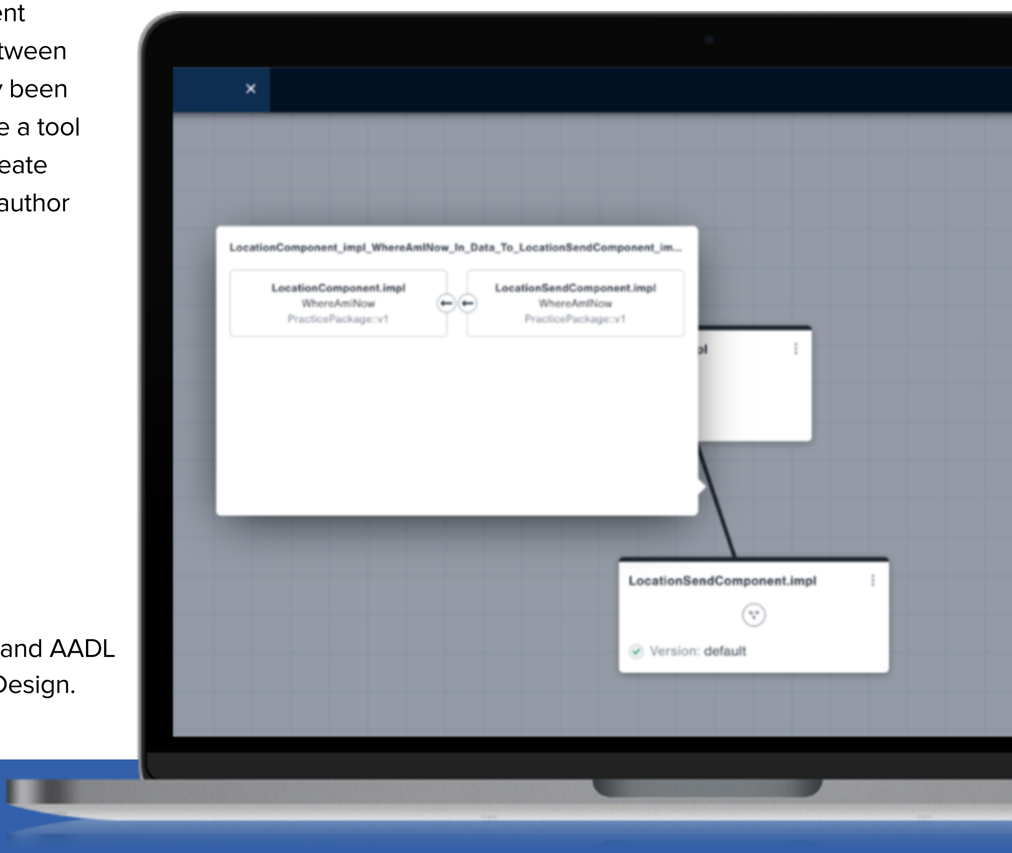
This screen capture shows a virtual UAV "talking" via an open standard (LMCP) to other components complying to that standard but developed independently of Flex.

MOSA is accelerated by Model Driven Development (MDD)

Model Driven Development (MDD) connects system models and software code. Connecting system models with code accelerates system analysis, code maintenance, cybersecurity improvements, and system certification.

Flex was designed as a MDD language. Interface descriptions can be shared between Tangram Pro and other MDD tools and languages, such as SysML or AADL (which are MDD languages for describing system architectures).

Flex was specifically designed for reuse and sharing of specifications. In many cases, relevant Flex specifications for component software interfaces, or for transforms between different interfaces, may have previously been defined. A system engineer can leverage a tool such as Tangram Pro to automatically create integration code without ever having to author Flex themselves.


LocationComponent_impl_WhereAmINow_In_Data_To_LocationSendComponent_im...
LocationComponent.impl
WhereAmINow
PracticePackage::v1
LocationSendComponent.impl
WhereAmINow
PracticePackage::v1
LocationSendComponent.impl
Version: default

Tangram Pro™ leverages Flex and AADL to provide this Model Driven Design.

KEY TERMINOLOGY

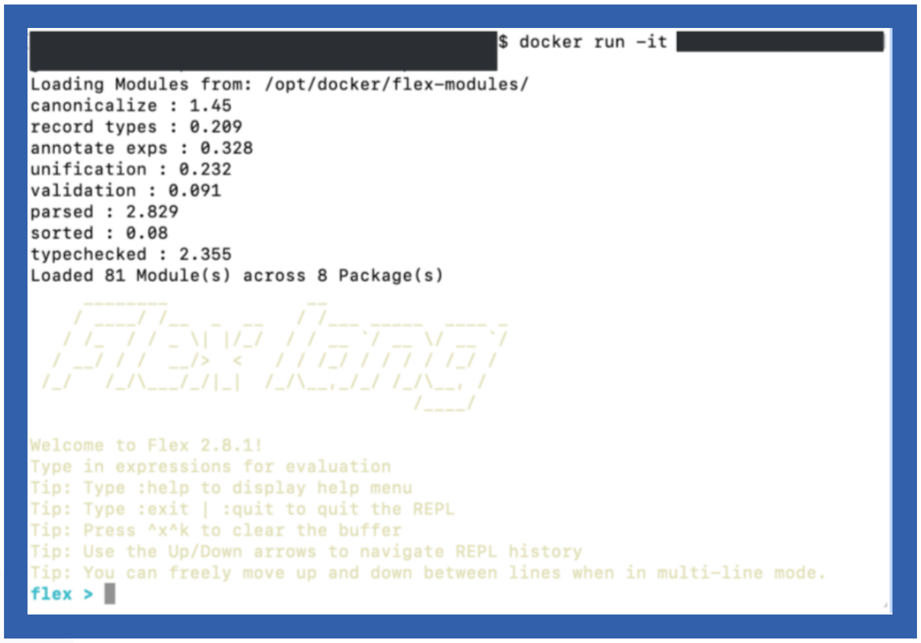
Model Based Systems Engineering (MBSE):
the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases. It is a systems engineering methodology that focuses on creating and exploiting domain models as the primary means of information exchange, rather than on document-based information exchange.

Model Driven Development (MDD):
A development paradigm that extends MBSE by using models as the primary artifact of the development process. Usually in MDD, the implementation is automatically generated from the models.

3c. Flex Authorship and Agile Development

Flex can be authored with cloud-based tools that align to agile processes

Flex comes with Integrated Development Environment (IDE) markup (such as syntax highlighting, type checking, and browsing), so that engineers can use their preferred IDE, such as VS Code or IntelliJ. Coming soon is a downloadable Docker image for a Flex read-eval-print loop (REPL) interpreter to provide immediate testing feedback on Flex specifications.

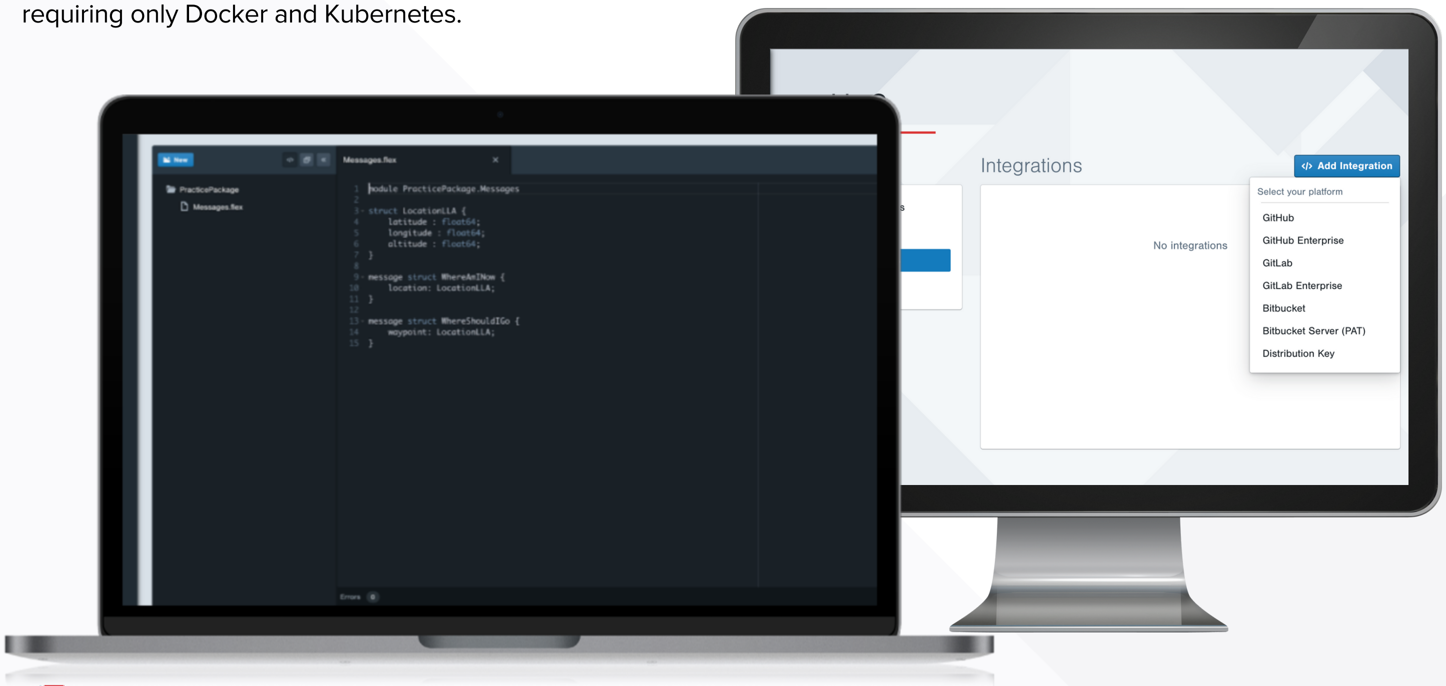


```
[redacted]$ docker run -it [redacted]
Loading Modules from: /opt/docker/flex-modules/
canonicalize : 1.45
record types : 0.209
annotate exps : 0.328
unification : 0.232
validation : 0.091
parsed : 2.829
sorted : 0.08
typechecked : 2.355
Loaded 81 Module(s) across 8 Package(s)
    ________              __
   / ____/ /__  _  __    / /___ _____  ____ _
  / /_  / / _ \| |/_/   / / __ `/ __ \/ __ `/
 / __/ / /  __/>  <    / / /_/ / / / / /_/ /
/_/   /_/\___/_/|_|   /_/\__,_/_/ /_/\__, /
                                    /____/

Welcome to Flex 2.8.1!
Type in expressions for evaluation
Tip: Type :help to display help menu
Tip: Type :exit | :quit to quit the REPL
Tip: Press ^x^k to clear the buffer
Tip: Use the Up/Down arrows to navigate REPL history
Tip: You can freely move up and down between lines when in multi-line mode.
flex > 
```

Flex is ideally suited for CI/CD and agile processes

Tangram Pro provides Flex authorship capabilities without any on-premise infrastructure requirements, and integrates with all popular CI/CD platforms. For secure settings, Tangram Pro is available as an on-premise install requiring only Docker and Kubernetes.



```
module PracticePackage.Messages

struct LocationLLA {
    latitude : float64;
    longitude : float64;
    altitude : float64;
}

message struct WhereAmINow {
    location: LocationLLA;
}

message struct WhereShouldIGo {
    waypoint: LocationLLA;
}
```

3d. Confidence in Interface and Transform Correctness

Flex-generated component software interfaces and transforms provide greater confidence and trust in correctness.

Flex feels like code, but it is actually a precise specification language. This means:

1. The meaning of any Flex specification is mathematically precise and unchanging, and is independent of things like execution speed, or time, or source code, or compiler, or computer architecture.
2. Flex specifications are amenable to a wide range of analysis techniques (from automated test generation though to formal methods), as is code auto-generated from Flex specifications.
3. Flex specifications have intrinsic algebraic properties that allow them to be fused, merged, and otherwise blended to create new composite specifications, or variant specifications in alternate forms.
4. Flex specifications can be used to automatically create secure and correct implementation code in a variety of source languages (including C++, Java, Rust) that conform to a variety of coding standards (including OMS, LMCP) and which leverage a variety of transport mechanisms (including DDS, ZeroMQ).

Flex specifications have strong algebraic properties

One key attribute of specification languages is that they are suitable for mathematical reasoning. To understand the significance of this, consider the following mathematical equations, translating the temperature in f in Fahrenheit to the temperature c in degrees Celsius:

$F = (9/5)C + 32$, and $C = (5/9)(F - 32)$

To check we got these equations (or translations) right, we can do algebra on them. We want to ensure that if we convert from Fahrenheit to Celsius and back again, we get what we started with.

$F = (9/5)C + 32$

$F = (9/5)((5/9)(F - 32)) + 32$ {substitute the equation for C}

$F = (F - 32) + 32$ {multiplications and divisions cancel}

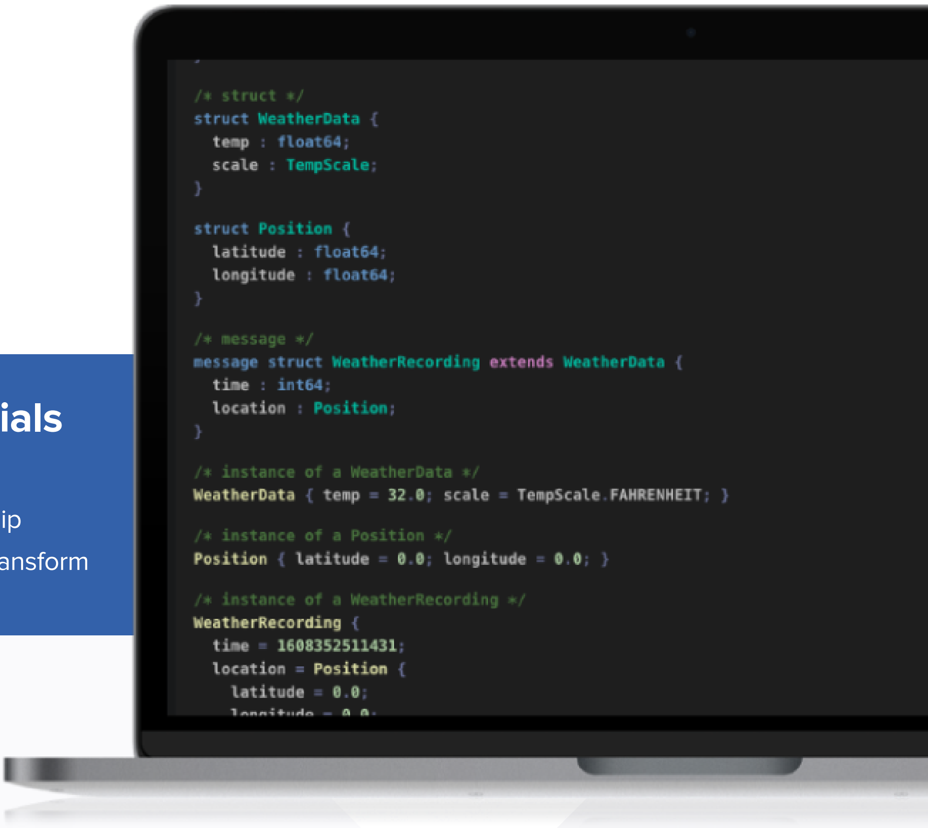
$F = F$ {additions and subtractions cancel}

Obviously this was a trivial example of mathematical reasoning. But the same reasoning principles apply to transforms between different kinds of messages. Even in very complex transformations, formal reasoning tools can provide answers to questions like:

- If I convert a message to XML and back again, will I always get the original message back?
- If I translate a message from latitude and longitude to a local coordinate system and then translate it back, what is my maximum error loss?
- If I transform a message into another format, can I guarantee that the transform won't introduce a value that will cause downstream problems?
- If I want to transform a message into a more limited format, can I find out what subset of messages it is guaranteed to work for?

This insistence on algebraic properties is influenced by the growing area of Formal Methods as a way to provide rigorous guarantees that whole classes of errors cannot arise.

It is no easy thing to design a rich message specification language that also allows for strong mathematical reasoning. Happily, Flex was designed by world-class computer scientists and system engineers to ensure that it possesses true algebraic properties while still able to handle real-world issues like noting the current time of a transform, or handling asynchronous message streams.



```
/* struct */
struct WeatherData {
  temp : float64;
  scale : TempScale;
}

struct Position {
  latitude : float64;
  longitude : float64;
}

/* message */
message struct WeatherRecording extends WeatherData {
  time : int64;
  location : Position;
}

/* instance of a WeatherData */
WeatherData { temp = 32.0; scale = TempScale.FAHRENHEIT; }

/* instance of a Position */
Position { latitude = 0.0; longitude = 0.0; }

/* instance of a WeatherRecording */
WeatherRecording {
  time = 1608352511431;
  location = Position {
    latitude = 0.0;
```

Find more Flex examples and tutorials

- https://docs.tangramflex.io/docs/flex/start
- https://docs.tangramflex.io/docs/tutorials/flex_authorship
- https://docs.tangramflex.io/docs/tutorials/implement_transform

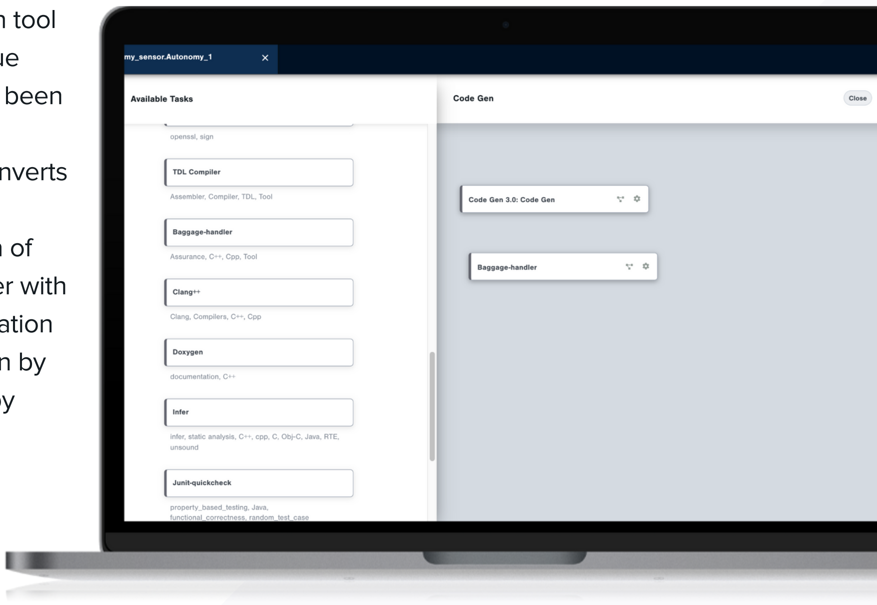
3e. Testing and Validation

Flex is ideally suited for rigorous testing and validation protocols

Because Flex is a true specification language (especially its mathematical and deterministic properties), it can automatically generate suites of validation tests that remain up-to-date even as the project progresses. These test suites provide coverage guarantees, and fit well within standard assurance and certification processes.

Flex also provides support for going deeper into more modern, emerging verification techniques that provide stronger guarantees for properties such as cyber resilience than can be obtained through standard assurance and certification processes. or example, the Baggage Handler symbolic-execution tool developed by the formal methods company Galois (https://www.galois.com) has been adapted to work with Flex. Baggage Handler can take a Flex-authored component interface (or transform) together with C++ code for that interface (or transform) and verify total correctness, i.e. that the C++ code behaves exactly as specified in the Flex specification, no more, no less. If Baggage Handler notes that the verification is not possible, it will provide a counter-example to show where corrections need to be made.

For another example, the Narcissus code-generation tool developed by MIT and Purdue (https://www.cs.purdue edu/homes/bendy/Narcissus/narcissus.pdf) has also been adapted to work with Flex. Narcissus takes a Flex specification of a serialization transform (one that converts a structured interface into a data stream like XML or binary), and automatically produces the specification of the corresponding deserialization transform, together with a mathematical proof of total correctness. Deserialization code is very hard to get absolutely right when written by hand, and many many network cyber-attacks work by exploiting vulnerabilities in deserialization code.


my_sensor.Autonomy_1
Available Tasks
openssl, sign
TDL Compiler
Assembler, Compiler, TDL, Tool
Baggage-handler
Assurance, C++, Cpp, Tool
Clang++
Clang, Compilers, C++, Cpp
Doxygen
documentation, C++
Infer
infer, static analysis, C++, cpp, C, Obj-C, Java, RTE, unsound
Junit-quickcheck
property_based_testing, Java, functional_correctness, random_test_case
Code Gen
Close
Code Gen 3.0: Code Gen
Baggage-handler

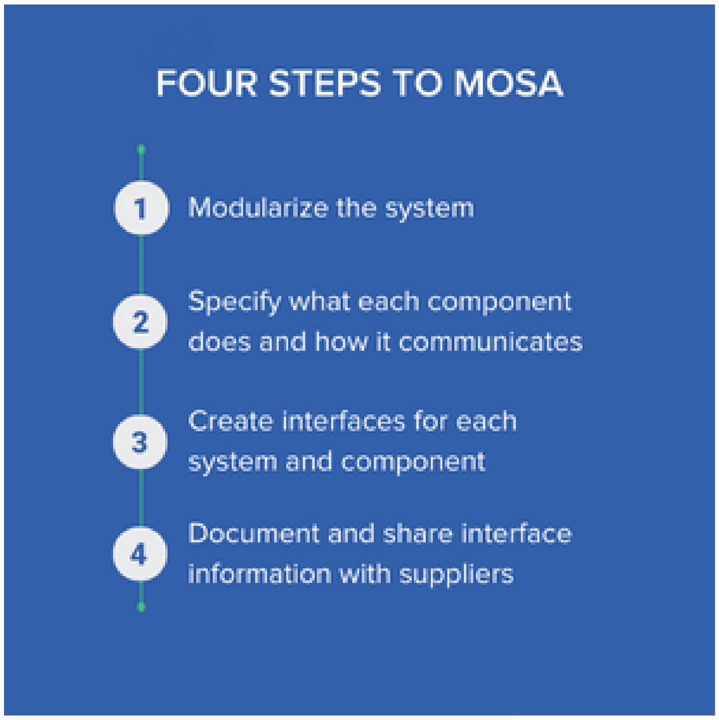
4. Concluding Notes

4a. Flex Language and the DoD MOSA Mandate

The 2021 National Defense Authorization Act (NDAA) requires the adoption of a modular open systems approach, including:

- Machine-readable definition of the relationship between the delivered interface and existing common standards or interfaces available in the interface repositories (to be established)

- JADC2 conduct demonstrations and complete an assessment of the technologies developed under the System of Systems Integration Technology and Experimentation program (STITCHES)



FOUR STEPS TO MOSA

1. Modularize the system
2. Specify what each component does and how it communicates
3. Create interfaces for each system and component
4. Document and share interface information with suppliers

Flex-like technologies are needed to realize this initiative.

Flex provides the capability to connect component interfaces to standards. This enables platforms employing Flex to create component interface repositories.

Flex is designed to be machine-readable, as shown by its use in Tangram Pro's automatic transform and component software interface capabilities. It is also human readable and well suited for model driven architectures and software development.

Flex was developed to bring STITCHES-like integration and HACMS-like trust capabilities to systems engineers and software developers of mission critical systems. STITCHES transforms can be converted to Flex as illustrated by a Tangram Flex converter tool that does just that.

4b. Flex and Tangram Pro™ Reference Material

The Flex Specification Language is freely available and is also adopted as part of the Tangram Pro™ Component Software Integration Platform (CSIP), an engineering platform for mission-critical systems that bridges the treacherous gap between system engineering and software development with a practical engineering workspace for assured component deployment.

To learn more about Flex and Tangram Pro™, visit our documentation site at docs.tangramflex.io

Additional reference materials are linked below:

The Business Case for a CSIP

Implementing Component Based Systems Engineering

Solving the Data Problem with Component Software Interfaces

Perspectives of a Systems Architect

Realizing DevSecOps in a Digital World

TANGRAMFLEX®

MARK STADTMUELLER
VP Product Management
mark.stadtmueller@tangramflex.com