GAPS-CLOSURE auto-generated xdcomms: system model and behavioral specification

Max Levatich

March 7, 2023

Contents

1 Introduction

TK.(one sentence)

1.1 About xdcomms and the GAPS-CLOSURE project

TK.

1.2 About this document

TK.

2 Summary

TK.(one sentence)

2.1 High-level overview of xdcomms auto-generation

TK.

2.2 Goals of xdcomms design and implementation

TK.

3 Information Model

In this section we formally model all of the datatypes and classes used in the xdcomms system, and provide intuition as to their purpose; the application control flow and function behavior is fully specified in Section 5.

3.1 Summary diagram

The following UML diagram provides a bird's eye summary of the information model. TK.

3.2 Primitives and Aliases

TK.

3.3 Cross-domain data format

This subsection describes the format of program data which can be transferred from one application endpoint to another via xdcomms, and the intermediate representations of said data.

3.3.1 The `Data` type

xdcomms uses a simple type, `Data`, to capture a collection of program data which is eligible to be transferred cross-domain to another application endpoint. It is isomorphic to a struct in C; the purpose is to *lift* valid, serializable structs into the xdcomms information model.

`Data` has two fields:

- `dtype : DataType` is a string identifier for this data format.
- `contents : [CValue]` is the ordered list of fields, where `CValue` refers to a typed value in C that is a scalar, vector, or serializable struct.

We restrict `CValue` to *serializable* datatypes, meaning a `CValue` is not a pointer, or, if it is a struct, the struct fields are all serializable (i.e. not pointers). Because application endpoints are physically and memory isolated, pointers cannot be transferred cross-domain. Fixed-length arrays, however, are valid `CValues` (because fixed-length arrays are stored by value in C structs).

3.3.2 The `Marshalled` type

TK.From function arguments to struct with trailer.

3.3.3 The `Serialized` type

Before being sent cross-domain, data converted into an in-memory `Marshalled` struct with a sequence number and error correction codes must additionally be:

- Tagged with a `GTag`.
- Given a unique ID to distinguish it from other incoming request or response packets at the destination.
- Coerced into the packet format expected at the destination device.
- Serialized into a stream of bytes in network order.

This transformation is mediated through the `Serialized` type.

The GAPS tag, or `GTag`, is a tuple of three tags: a `MuxTag` identifying the source and destination applications, a `SecTag` identifying the security levels between which the data is being transferred, and a `TypTag` encoding the `DataType` as an integer (so that it can be unmarshalled at the destination). So for `GTag` we have the type `GTag : (MuxTag, SecTag, TypTag)`.

TK.IDs, packet format, bytes, network ordering.

Conversion between a `Marshalled` struct and its corresponding `Serialized` struct is mediated by the `serialize()` and `deserialize()` functions in the device-aware `Codec` class (Section 3.4.3).

3.4 The hardware interface

TK.(describe subsection)

3.4.1 The `HALConfig` class

TK.

3.4.2 The `Device` class

TK.

3.4.3 The `Codec` class

TK.

3.5 The network abstraction

TK.(describe subsection)

3.5.1 The `Binding` type

TK.

3.5.2 The `XDContext` type

TK.

3.5.3 The `Wrapper` class

TK.

3.5.4 The `Handler` class

TK.

4 Application model

TK.(describe section)

4.1 The `MasterSequence` class

TK.

4.2 The `EventQueue` class

TK.

4.3 The `RPCTransaction` class

TK.

4.4 The `HAL` class

TK.

4.5 The `App` class

TK.

4.6 The `AppThread`

TK.

5 Behavioral specification

TK.(describe section)

5.1 Control flow

TK.(should include both an english description and detailed diagrams)

5.2 Function-level contracts

5.2.1 `Wrapper.marshall()`

The `marshall()` function is called by the `Wrapper` to marshall the incoming arguments (the arguments to the cross- domain function in the original, unpartitioned program) into one serializable data structure with error correction and a sequence number.

Arguments: `d: Data, req_counter: int` (`d` is unpacked and may represent multiple or no arguments)

Return value: `m: Marshalled`

Pre-conditions:

- The value of `req_counter` is either non-negative or `INT_MIN`.
- The `Marshalled` struct defines, in the same order as the arguments to `marshall()`, one field of the same type and name as each argument, followed by a trailer datatype.

Post-conditions:

- For each incoming argument, the corresponding field in `m` is set to the value of that argument.
- The `trailer` of `m` has its `seq` field set to the value of `req_counter`.

6 Generator operation

TK.(describe section)

6.1 Inputs: The GEDL

TK.

6.2 Outputs

TK.

7 Whole-system correctness properties and proofs

TK.(describe section)