Common API C

Project Purpose

The purpose of this project is to enable programs written in C to work with interfaces defined in Franca IDL. This would allow writing application code for both clients and servers that does not depend on the communication mechanism between them (e.g., D-Bus, SOME/IP, in-process, etc.) For this to work, the application code needs to rely on a defined mapping of Franca IDL to C language constructs and on the corresponding run-time support. Additionally to that, the communication via a particular mechanism is implemented by the binding code that is generated automatically.

The following principles and constraints apply:

- Align as much as possible with the Franca IDL mapping (e.g., for the data types) and implementation features (e.g., the approach to concurrency) implemented by Common API C++.
- Rely on the existing Franca framework for model transformations and code generation under Eclipse.
- Leave with applications the design choices related to concurrency (i.e., the main event loop vs. threading), to memory management (i.e., dynamic vs. static allocation) and to other major areas.
- Prioritize D-Bus/kdbus and in-process communication over other mechanisms for Linux environments.
- Support non-Linux environments and especially embedded, resource-constrained systems (e.g., do not require using dynamically allocated memory).
- Long-term, minimize the redundancy with the Common API C++ in the areas of Eclipse tooling and run-time support (e.g., backend libraries).

Usage Scenarios

The following usage scenarios are envisioned for Common API C. They are simultaneously the needs that provide the motivation for project.

- A small, low overhead system component provides some services via IPC. The code is written in C with as few dependencies as possible. The specific choice of IPC needs to remain flexible.

- A service or application component consumes system services via IPC. The component is intended for usage in different software platforms that include GENIVI-compliant ones as well as less sophisticated environments. The specific choice of IPC needs to remain flexible.

Mapping Franca to C

The goals (possibly conflicting):

- Enable as many Franca IDL features (e.g. inheritance, namespaces, etc.) as possible.
  Rationale: What is possible with other languages (e.g. C++) should be possible in C as well.
- Keep the language mapping (e.g. data types, etc.) as close as possible to C++.
  Rationale: Since C is [mostly] a subset of C++, switching between two languages should bring as few surprises as possible.
- Keep the overhead (e.g. start-up time, CPU load, etc.) induced by the mapping as low as possible.
  Rationale: Lower overhead is most likely the reason why someone would choose C over C++.
- Mapping of identifiers (e.g. the usage of case and underscores, prefixes, etc.) and indentation (tabs vs. spaces and width) in the generated code is configurable.
  Rationale: Generated bindings that match the style of the their host project reduce the "impedance mismatch".

Comparison of Franca IDL Mappings is the working draft of the mapping.

Infrastructure

The following Wiki pages are used to distill and document the project requirements:

- Open questions brought up for discussion
- Formal requirements identified for the project

Any issues and concerns that lack an agreed approach to address them are tracked as open questions. Once a proper resolution has been proposed, the resulting requirements are extracted and documented. The requirements go through the usual review and approval by the Expert Group.

All discussion related to the Common API C project should take place on the public GENIVI mailing list for IPC:

- genivi-ipc@lists.genivi.org

Proof of concept implementation is hosted on public GENIVI git:

- github.com/GENIVI/capic-poc

Bug tracker is hosted on public GENIVI JIRA

- capic poc tracker.

Related Work

The following projects are related to Common API C:

- Franca
  - Franca binaries and release history
  - Xtend / Xtext documentation
  - Eclipse Modeling Framework (EMF) documentation
  - D-Bus EMF model (along with Eclipse update site)
- Common API C++
  - Common API C++ Public Wiki
- SOME/IP back end for Common API C++
- ipc-quartztime
- Code generator suite for Franca IDL that does not rely on Eclipse platform and Java. It is runnable from the command line and produces C server stubs and proxies using the GDBus library.
- CORBA mapping for C language
- Apache Celix
  Apache Celix is an implementation of the OSGi specification adapted to C. It will follow the API as close as possible, but since the OSGi specification is written primarily for Java, there will be differences (Java is OO, C is procedural). An important aspect of the implementation is interoperability between Java and C. This interoperability is achieved by porting and implementing the Remote Services specification in Celix.
- Apache Thrift
  Apache Thrift is an interface definition language and binary communication protocol that is used to define and create services for numerous languages.
- ZeroC Ice is built around the Internet Communication Engine (Ice). The latter is a modern object-oriented toolkit that enables one to build distributed applications with minimal effort. It comes with its own IDL – Slice – as the fundamental abstraction mechanism for separating object interfaces from their implementations. No bindings for C are supported, though.

The following projects could be used by Common API C as run-time dependencies:

- libsystemd
  Includes D-Bus / kdbus bindings (sd-bus) and a generic main event loop implementation (sd-event).
  - LWN discussion of the D-Bus bindings
  - Lennart Poettering's blog on sd-bus API
- Embedded Linux Library (ELL)
  Includes D-Bus / kdbus bindings, main loop implementation, string utilities, etc.

The following libraries implement complex data types in C:

- GLib
  Hash maps, arrays, ...
- Apache Portable Runtime (APR)
  Hash maps, arrays, ...
- Embedded Linux Library (ELL)
  Hash maps, bit sets, ...

The following projects implement messaging via different transports including in-process:

- ZeroMQ messaging library written in C++.
- nanomsg messaging library written in C.

The following efforts target serialization/deserialization of parameters:

- GVariant serialization format is used by kdbus/dbus2 to make access to serialized message content more efficient.
- Protocol Buffers is Google's implementation in C++.
  Several C implementations of the Protocol Buffers specification are available:
  - nanopb (see also the repository and documentation); targets low memory footprint and avoids dynamic memory allocation
  - protobuf-c (somewhat obsolete–last commit in Mar 2015); used to support more specification features while being less 'embedded'
- MessagePack claims to be an "efficient binary serialization format" that "lets you exchange data among multiple languages like JSON", but "faster and smaller". Implementation in C is available.
- BSON claims to be "bin-ary-en-coded seri-al-iz-a-tion of JSON-like doc-u-ments". Implementation in C is available.

The following efforts are related to IPC benchmarking:

- D-Bus vs. kdbus, see LKML, 2015-04-25
  Describes the setup and the results of using GLib bindings with kdbus vs. D-Bus as the backend.

The following libraries implement a main event loop:

- GLib
  Implements a main event loop among a whole lot of other things.
  - GLib documentation
  - GLib main event loop documentation

- sd-event
  Implements a main event loop as a part of libsystemd. Includes event source prioritization, optimized timer event management along with signal and child handling on top of `epoll`.
- libev
  Tries to do one thing only (POSIX event library), and this in the most efficient way possible.
  - libev documentation
- libuv
  Provides a cross-platform implementation of event loop along with asynchronous I/O, threading utilities, etc. Replaced libev in the node.js implementation.
  - Code repository
  - Book about libuv

The following projects are related to parsing, code generation and analysis:

- Boost Spirit
  Provides a set of C++ libraries for parsing and output generation implemented as Domain Specific Embedded Languages (DSEL).
- libclang
  The C Interface to Clang that provides a relatively small API exposing facilities for parsing source code into an abstract syntax tree (AST), traversing the AST, etc.

Minutes and Presentations

- 2016-04-28, Working Session at AMM in Paris
- 2016-04-28, Presentation at AMM in Paris
- 2015-10-21, Presentation at AMM in Seoul
- 2015-04-23, Working Session at AMM in Stuttgart