edantech

ACP245 testing with automated tools

White paper

By Brazil's regulation, the ACP245 protocol must be included on every vehicle-installed Telematic Control Unit (TCU) and supported by every Service Operator (SO) allowed to operate on the country. This requirement has the specific purpose of avoiding vendor lock-in, by making sure that every TCU is able to communicate with every SO. In this way, if a consumer wants to change from one SO to another, or change her vehicle while maintaining the same SO, he can do it without worrying about support issues.

However, protocol compatibility has always been a problem. When multiple implementations of the same protocol come to exist and each tries to connect to the other, problems can arise because of *different interpretations* of its specification, protocol *flexibility and complexity*, which may allow for different ways to frame a message, and because of *selective implementation* that covers only the items used by a specific TCU.

ACP245 is not free of these problems, and it can be argued that, because of its origin and genesis, they are even accentuated. For example, consider the following points:

It was designed for extensibility, by using different protocol versions

The protocol includes a version field which defines the version of the protocol used by a TCU. This field is included so that new features can be added without needing to update every single TCU already deployed.

However, to guarantee that a TCU can be used with any SO, every SO needs to support every version, even versions that are no longer included on new TCU but that may be deployed on older vehicles. So, even if a SO works with a given number of TCU, older or newer devices may fail to work because of protocol version differences.

On the current specification, there are already three different version numbers, and an extension has been added to extend possible version numbers from 7 to 32. The first two won't probably be present on production vehicles, but this is anyway an indication that the field will be used, and that new version will probably appear after the first TCUs are deployed.

It was designed for extensibility, by using predefined extension flags

Even while using the same protocol version, some messages may include information that is not defined by the ACP245 specification, but is particular to each manufacturer.

Implementations should ignore items they can't understand, but, most of the time, they are not tested to cover every possible case in which a message is extended, since those are, by definition, almost infinite. Therefore, a SO that has been tested against a wide range of devices can fail when tested with a new one, because the it uses a particular extension which was not covered by the SO implementation tests.

Also, since ACP245 is a binary protocol, simple errors on bit packing may be hard to identify, since the messages may be accepted as valid but parsed incorrectly (for example, by assuming that a section damaged by the bit packing error was actually an extension). On

the other hand, errors that may be ignored by one implementation because they are interpreted as an unsupported extension, may affect implementations that do use that extension.

It has options on how to represent the same information

For example:

- The TCU Serial number, and other optional elements, can be represented as a string, a packed decimal, or as binary data. Each TCU can represent this field using any of those options.
- An SO may, or may not, send a transmit interval field which requires a TCU to change its report interval at the same times it performs another operation on the TCU (for example, blocking). Functionally, a similar effect can be achieved by sending two independent messages, one with the command, and the other with the new configuration.

Different options mean that a TCU or SO may be fully compliant, since all protocol functions can be requested and processed, but not interoperable, since a different way to frame a message, or the inclusion of optional elements, could cause the implementation to fail.

Its specification includes sections that lead to an ambiguous interpretation

For example:

- The meaning of values 3 and 4 of the Transmit Interval field are not precisely defined.
- The vehicle descriptor element can include fields that are not defined, like the vehicle color, the license plate, etc.. One TCU may send some of these elements, others may not.
- The satellite information on the Location Element includes a field called “Number of satellites”, and a list of satellites ID. During the protocol definition stage, some implementors assumed that the “Number of satellites” was the number of included satellites ID, but other didn't, and assumed it was an independent field. This is reflected on the example messages, which include an 8 on the number of satellites, but no satellite IDs. The specification is not clear on this issue, and both interpretations are valid.

Ambiguous elements are, by definition, a point of failure for interoperability, since different implementations may act differently upon reception of those messages.

All this protocol flexibility generates different ACP245 protocol *dialects*, each using different subsets of the protocol to communicate with its peer. A 100% correct implementation, *must* be able to handle every dialect (excluding undefined or ambiguous items, which will always cause problems). However, there are no guarantees, whatsoever, that if one dialect is supported by an implementation then it will be able to inter-operate with another. This raises the importance to perform *very careful* testing to guarantee interoperability.

Its possible that all this flexibility and extensibility won't be used in the future, but the protocol on which ACP245 was based[1] included it, so ACP245 includes it, and so it must be handled by each implementation.

Similar telematics initiatives, like the European *E.Call* system, have elaborated a process for performing conformance and interoperability test suites, that check that: all the functions of a device can be exercised by using the selected protocol, and that different implementation of the

1 A protocol from Motorola, called ACP 3.0.1.1.

protocol can inter-operate to requests these functions. This process is driven by certification entities, like for example, the GST-Certecs project[2], which was responsible for documenting this process for the *E.Call* system.

On this scenario, it's our opinion, there are only two options to ensure 100% conformance and interoperability between TCUs and SOs. Both options require a central certification entity which either:

a) performs conformance testing of TCUs and SOs using a specific subset of the ACP245 protocol, which must cover all the functional aspects of the Brazil regulation, and therefore ensuring that every TCU and every SO are able to interact by using that specific subset.

b) or, builds and maintains a centralized repository of different protocol dialects, one for each TCU and SO, and tests each TCU against each SO protocol dialect, and *vice versa*.

Both options have its pros and cons:

- Option a), limits ACP245 extensibility, since a TCU must use the subset defined by the certification authority to communicate with a given SO. However, TCUs/SOs may use an extended protocol when connecting to specific SOs/TCUs. This options would also require explicit support from the entities defining ACP245, since there's nothing in the specification that requires that a specific subset is used.
- Option b), requires each SO and TCU to have a complete protocol implementation (which should be the case), not only to implement a minimum set so every function can be invoked, but to allow each function to be invoked with many variations of similar messages. Also, the certification authority needs to have a programmable and flexible ACP245 test suite that allows it to create, parse and interpret, different messages, and execute all the possible interactions against the TCU or SO under tests without much effort.

To allow repeatable and supervised tests, an automation tool must be used to support this conformance and interoperability test process. Otherwise, if a test can't be performed *exactly* as the certification entity requires, conformance, in the sense of testing that all functions can be invoked by means of the protocol, may be checked, although questionably, by doing *ad hoc* tests, but interoperability will be, in our opinion, unreachable.

Edantech approach to protocol validation

We believe that the best approach is to merge ideas from these two options, defining a comprehensive test-set that exercises the different message flows using message frame variations. These tests should not reflect a particular TCU or SO operation, and should instead be a sample of valid interactions defined by the specification.

For example, to block a vehicle, different types of messages could be sent on each test case, one including an IMEI, another with some optional components, one changing the timer interval on the same message, etc.. Also, different protocol version can be exercised by running a different test-set.

Even if passing all these tests will never guarantee a 100% protocol conformance, it will be a good indicator that every function can be performed by the protocol, and that the implementation is able to handle multiple message variations, and therefore, will probably handle others not covered by the tests cases. So, if a TCU connects to a SO that uses a particular variation of messages, or a SO must support a new TCU which uses some extension fields, there would be a reasonable certainty that

2 http://www.gstforum.org/download/White%20Papers/DOC_CERTECS_White_Paper.pdf

they will inter-operate.

This test repository should be easy to modify, to integrate new tests, to add new variations of certain messages, and also to easily understand each test by looking at its definition. It should also be possible to validate test consistency, by executing each TCU test against the opposite SO test, making sure that the tests themselves do not include errors.

It also desirable that tests progress can easily be monitored and stored, so that discrepancies on protocol implementation can be spotted and communicated effectively to each implementor.

In this context, we believe that Edantech ACP245 Test Suite and its ACP245 script language[3] is an appropriate tool to build this test cases database, since it was built with these ideas in mind. We base this opinion on the following:

- ✔ The Test Suite can be used as an ACP245 server or client, so that establishing a connection to a SO or receiving one from a TCU can be performed with the same tool, on the same testing environment, against the same ACP245 implementation.
- ✔ The ACP245 implementation has been designed to be 100% complete. We didn't skip sections that we didn't use.
- ✔ The script language makes it easy to define new tests, to modify previous tests, and also it's easy to understand, even for non-programmers. The language is designed to write test cases, by making assertions, and checking conditions and timing issues.
- ✔ All tests results are stored for later analysis, and binary messages are decoded to check that their content is as expected.

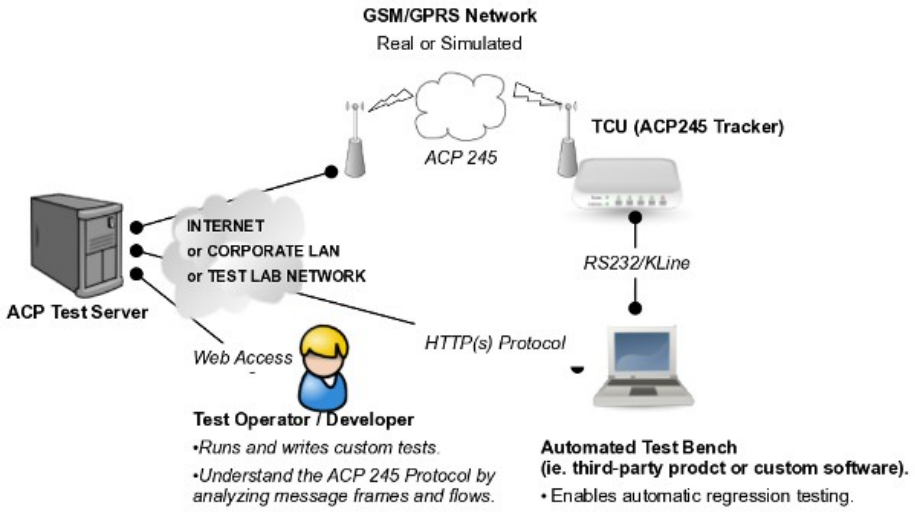

Figure 1: Edantech ACP245 Test Server used for TCU validation

To perform a test, a script should be written using a set of messages that represent a valid ACP245 message flow. These messages will be one of a number of possible variations supported by the protocol. The script can be stored, and later executed against the system under test. Test results and message flow, are stored for later review. On all these interactions, no manual operations are performed, tests are always executed following the script, in a repeatable fashion, in which controlled changes can be performed, stored, reviewed, and replayed.

3 http://www.edantech.com/solutions/products/acp245-test-server

During a test, validations should also be performed on the SO and the anti-theft device, to check that the requested actions were performed. These tests could be performed manually, but, ideally, at least for the TCU side, they should be integrated with the tool and tested using the script language. Since Edantech ACP245 library is already written in C and used from the high-level script through language bindings, it's possible to create bindings to another C library to perform this input/output checks from the test scripts without much effort, centralizing all the test knowledge on the script database.

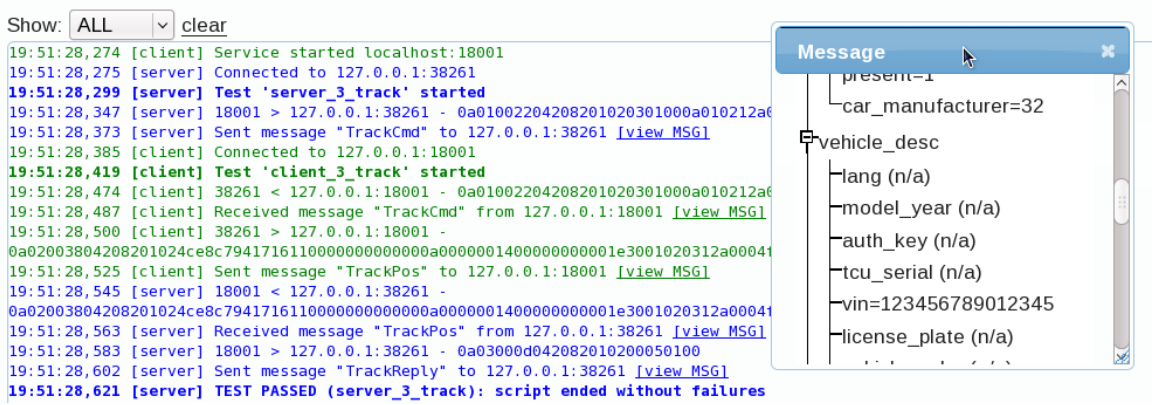

Figure 2: Console Log Panel

These tools are not a definitive answer for all the interoperability problems that we are going to face with ACP245. Interoperability and conformance testing, will always be a complex subject in which different answer will provide different solutions, with it's pros and cons. But whichever is the selected solution to face this problem, we must always aim for repeatability and control on the test procedures, and remember that implementation independence is one of the main objectives.

EDANTECH (ILWICK S.A.),

Av. Libertador 1807,

11800, Montevideo, URUGUAY

Phone#: +598 2 929 0029

Fax#: +598 2 924 8013

e-mail: info@edantech.com