Service Oriented Architecture: impacts and challenges of an architecture paradigm change

Marc Bellanger, Edward Marmounier

Service Oriented Architecture: impacts and challenges of an architecture paradigm change

Regular Paper

Reference: Paper 78

Authors: Marc BELLANGER and Edward MARMOUNIER (RENAULT SW labs)

1 Abstract

Automotive embedded software relies on signal-based architecture for a long time. This architecture has proven through the last decades its reliability and ability to address complex systems such as a car embedding several tens of processors.

Automotive industry foresees a large introduction of Service Oriented Architecture in the car whereas the technology was initially used by information systems and web applications. A complete change of architecture is clearly a challenge considering the number of heterogeneous actors, the heavy legacy of business, the safety constraints.

This paper aims at providing feedbacks on the introduction of SOA in automotive industry through the prism of Software architecture and development team.

2 Problem statement

The SOA paradigm introduction can be considered as a big bang for the automotive industry, nevertheless the trend seems clear. The challenge of introducing SOA can be assessed through multiple questions:

- What are the main SOA concepts to adopt, challenge or ignore in the context of automotive?

- How an automotive company organization can digest this change by adapting the process accordingly?

- How to assess the actual benefit in term of time to market improvement for new features?

- How to guarantee the same level of safety and reliability of the overall system?

3 Main SOA concepts in Automotive

With cloud technologies growing, SOA is now used in most high availability infrastructures and it is adopted to enable inter-cluster communications where the network is seen as a resource to spread the workload across computation units. The SOA has proven a good flexibility to implement functional services, decoupled from each other, deployed in an heterogenous environment while de-risking the integration of all services together.

In automotive, software is becoming more and more important because the added value is no longer only in the mechanics of the car but now also in the growing feature set brought to the driver and its passengers such as driving assistance, entertainment, remote control, data reporting and many others. This wide area of software features shall be supported and maintained during the life cycle of the vehicle which is greater than 10 years. It means the software must be upgradable during this lifespan to prevent the car from losing value due to software feature obsolescence.

A wide feature set, with a long maintenance period, combined with the multiplicity of the ECUs (Embedded Control Unit) in the car, all require a new paradigm to architecture the car software. In this matter, SOA is powerful

As SOA comes with a dozen of principles, let's focus on some of the key concepts for automotive challenges and how Renault adopts or not these concepts.

3.1 Adopted SOA concepts

3.1.1 Contract based interfaces

The software contract must be a technical unit explicitly defining the functions or information a service shall provide, in which each input and output is explicitly described with a name and a type. The list of functions of the contract is called an interface. A service realizing an interface shall honor each function of the interface. A client using this interface, has the guaranty each function consumed will be honored by the service. A tool chain coming with an interpreter of the contract unit and a compiler for the client and service targets guaranty the contract is honored by the service.

This concept fulfils some requirements of the automotive industry where many actors are engaged and where the split of responsibility must be clearly defined to avoid misunderstanding. Moreover, with the rationalization of the number of ECU in the car, the functional split is not only at the boundary of the ECU but more and more inside the ECUs. With signal-based approach, the "contract" to formalize the shared information (signal) is often coupled to protocol used (CAN, LIN). By up leveling the contract to the actual need (which can be expressed by human) instead of a collection of unitary information with less context, it helps the mutual understanding of clients and service providers.

We will see later in the document that default interface concept proposed by SOA is not enough for automotive usage where data accurate description and usage periodicity (among other aspects) are important meta-data to be added to the contract.

3.1.2 Interface versioning

During the lifespan of the vehicle software maintenance, contract-based interface will evolve to follow the feature enhancements. It is critical to never alter a contract once it is already deployed and consumed by a client ECU, otherwise ECUs cannot be upgraded individually and the whole car system must be deployed within one baseline of contracts. This is not manageable due the number of ECUs in the vehicle. Instead, an interface shall be versioned following the Semantic Versioning[(1)] principles. The service realizing the interface can do it for one version or many versions depending on the development cycle of the service. On client side, it is recommended but not mandatory that it uses an interface in one version. The more there are versions supported by the client, the less impact we have on client ECUs but at the cost of maintaining old versions of a contract. Managing compatibility between clients, contracts and services is a matter of traceability matrix that shall be maintained up to date when deploying new software releases to ECUs

Service/version	Update Match?	Client/
modem.svc/2.0	Yes	modem.cli/[1.0 – 3.0]
remote_command.svc/1.3	Yes	remote_command.cli[>= 1.0]
car_data.svc/1.0	No	car_data.cli[>= 2.0]

Figure 1: Client and service compatibility matrix

Keeping an history of each component version for each vehicle is a mandatory requirement to allow the maintenance of such compatibility matrix, and it comes with a cost for architecture and design phase to handle the growing complexity of the system. Nevertheless, resolving this complexity at software execution time without versioning would be prone to errors that could not be recovered at run-time. (see **Error! Reference source not found.** for further information)

3.1.3 Transport abstraction

The SOA technology often comes with an abstraction (the binder) of the underlying transport protocol. It can be either TCP, UDP or Unix-domain socket, the developer does not have to care about how the interface will be called remotely or locally. IP addresses, ports of the services are handled by configuring the binder at integration time.

In the Automotive context, the time latency is very critical. The timing of data transmission is clearly dependent on the chosen binder technology. In Automotive, the trend is to use automotive grade ethernet. If the service must be realized within a specific timing constraint, the constraint must be clearly expressed by the client. Once we know the worst case of network usage (combination of all services used at the same time), it is possible to assess the delivery time of a service and check if it matches the performance requirement. The usage of QOS handling at ethernet level (802.1Q) helps guaranteeing the timing constraints for most sensitive services.

Another difference between signal based and service-oriented architecture is the abstraction level. Signal based architecture defines the data with the prism of data format on the wire. SOA concentrates on defining the actual service itself via a contract (IDL : Interface Definition Language) and not the way it is transported. The IDL associated to an SOA toolset offers the developers means to call methods, retrieve unitary or complex data with very few lines of code. The only format to care about is the format of the implementation data types. For example, a service providing GPS information can be seen by a developer as a method getLocation() returning a structure "location" combining "longitude", "latitude" and "elevation". From a developer standpoint, there is no need to care about the serialization format of the combined structure. Moreover, in SOA, the generated code will be the same for all clients using the structure. In signal based, it is likely to recreate a proprietary abstraction layer in every ECU consuming the data.

3.2 Challenging dynamicity in SOA

Service discovery covers multiple functionalities if we refer to SOMEIP-SD requirements[6] : Locate service instances, detect if a service instance is running, implement the publish/subscribe handling.

Do Automotive systems need these concepts?

Locate service instance is interesting to dynamically move services from one ECU to another during the car life cycle. However, when the services are assigned to safety functions, time critical functions, or highly reliable functions, there is a pushback on this dynamic allocation with regards to safety and security. The isolation of the functions into dedicated virtual networks brings a certain level of security protection. If the network configuration is static, it is more difficult for a hacker to get access to these virtual networks and the underlying assets (attack surface gets bigger in case of reconfiguration facilities in the car). Moreover, the service location discovery can take time at the initial usage of the service by a client. This is not compatible with features that need low latency responses. (even if some studies have shown that the service discovery sequence can last only few milliseconds [5]). A last point, the static configuration of service deployment allows to get a deterministic usage of network bandwidth. Today, the experience shows that deployment configuration remains static in the car. Nevertheless, a tradeoff could be to isolate the secured subnetworks from an "exposed" subnetwork where the dynamic allocation could be allowed (for infotainment for instance). Of course, the dynamic discovery crossing the subnetworks boundary would be forbidden.

The detection of service instance running is useful to implement a first level of health monitoring. SOA presents weakness compared to signal-based architecture in term of reliability of the information. Getting periodical heartbeat of the remote service provides to the clients a guarantee that the service host is alive, and the communication link is available. Of course, that does not guarantee the actual service can be provided when requested (for link congestion for instance). In addition to the health monitoring the service instance running detection can simply be a way to collect the information of the capability of the car. For example, for after sales options (HW or SW), the new part (ECU) or the new SW features could offer the service once installed. The infotainment could search/find for the service at startup and behaves accordingly. The service presence is decoupled from the static or dynamic allocation of the deployment parameters (mainly IP addresses)

Publish/subscribe concept relies on the service discovery as soon as dynamic subscription to an information is required. The dynamicity is one of the advantages of SOA compared to Signal-based architecture. In other terms, all clients are not running all the time, or a single client may need different information depending on the context. This SOA concept is massively used in the automotive.

In conclusion, publish/subscribe and service instance running detection are clearly in line with automotive expectations. For the dynamic location of service, even if the benefit in term of flexibility is clear, some security and reliability aspects must be considered before adopting the concept.

4 What are the challenges and how Renault addresses it?

4.1 Selecting the right SOA framework for automotive

Some common criteria to evaluate various SOA frameworks:

- Market adoption, licensing
- Implementation language support

- Supported OS
- Security
  In addition, automotive context brought extra constraints such as latency, network topology and resource usage.

Studied frameworks were: gRPC, JSON RPC, Thift, AutoSAR™ ARA:COM, Genivi CommonAPI with SOME/IP, RVI and Android AIDL / HIDL.

In the following chapters, we will elaborate on the solution chosen by RENAULT and the remaining open points we identified.

4.2 Heterogeneous ecosystems

The challenge to select and adopt an SOA solution is to ensure the compatibility with the multiple eco-systems present in the car. Among the most popular eco-systems, we can list the Classic and Adaptive AUTOSAR™, ANDROID OS or GENIVI™ GDP. Moreover, the adopted solution shall be ideally "standardized" and "well deployed" in the automotive industry.

4.2.1 AUTOSAR/GENIVI compatibility

4.2.1.1 Reference IDL (Interface Definition Language)

Renault has chosen one of the most popular and most flexible Interface Definition Language (FRANCA IDL[2]) as a reference and stable starting point. The choice has been motivated by the adoption rate of FRANCA in automotive industry, the readability of the file, the feature coverage, the open source availability.

It is important that any service developer or service user in the company (and stakeholders) talk the same language. That's why we push FIDL to be the only IDL language used. Even if some other SOA solutions (coming with their own IDL) will be used inside the car, we must stick using a single IDL format as a starting point. The conversion of IDL can be ensured using tooling. (see next chapter)

4.2.1.2 SOA middleware compatibility

There is no comprehensive solution today available on the shelf allowing to have a single and unique SOA solution whatever the OS used. Renault participates to standardization consortium (AUTOSAR™) and/or open-source alliance (GENIVI GPRO) to expose the problem statement of SOA compatibility across multi-OS.

RENAULT contributed to define

-FIDL usage guidelines[3][4] and

-IDL convertor tool (FIDL <-> ARXML)

to ensure the compatibility between AUTOSAR ARA::COM and GENIVI COMMONAPI. We can notice that ARA::COM and COMMONAPI are middleware layers of the SOA solution. Both middlewares use SOME/IP as communication standard. SOME/IP provides messaging element supporting the method based and event-based communication. The middleware proposes through the IDL, an upper level of interface abstraction but eventually the compatibility of communication is ensured via the SOME/IP standard. SOME/IP is far the most adopted standard for SOA in the automotive industry. We can also notice DDS but with less adoption rate today. See Figure 2

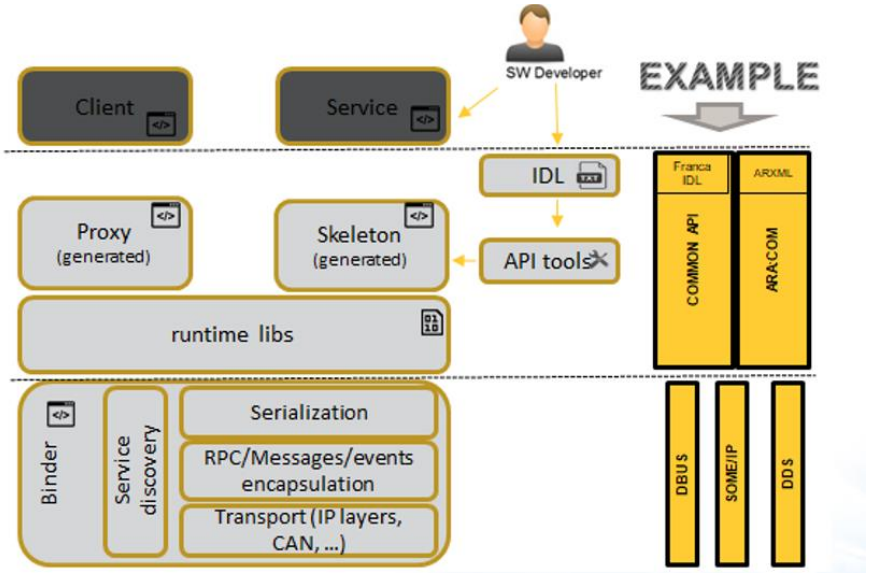

Figure 2: Presentation of SOA workflow and layers with application in GENIVI and AUTOSAR (extract from GENIVI AMM 2019 presentation [3])

4.2.2 ANDROID specific case

We talked about compatibility between COMMONAPI based ECU and AUTOSAR ECU previously but another important OS to take in account is ANDROID. By default, ANDROID does not support COMMONAPI natively and even worst, the IDL proposed by ANDROID (HIDL and AIDL) is not directly compatible with the FRANCA IDL. Moreover, the service discovery mechanism is very different from SOME/IP discovery. The services and clients are bound statically and internally in the ECUs. Exposing a remote (inter-ECU) service is like getting access to HW resources and ANDROID has a middleware layer to abstract the usage of HW resources (see Figure 3). Then, below the middleware, we can consider using SOA to access vehicle resources but not end to end.

As per our understanding, ANDROID can use the SOA concept but without a direct access between client and service. A middleware layer (acting as a proxy) shall be introduced to match ANDROID design constraints. The level of SOA integration within ANDROID is not clear today. Shall we keep the SOA concept below the HAL(hardware abstraction layer) or find a solution to expose SOA at application layer ?

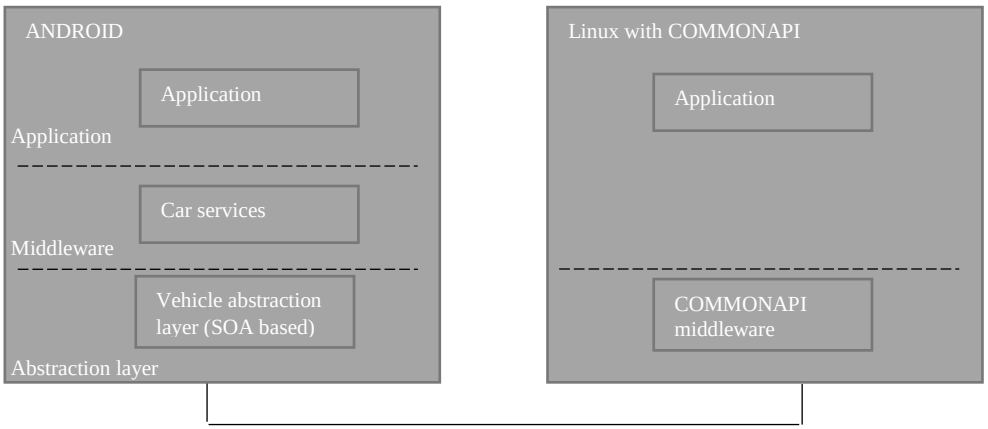

Figure 3 : ANDROID versus Linux COMMONAPI API based SOA implementation

4.3 SOA impact on the organization

Adoption of SOA paradigm has also impact in the Renault team organization. SOA workflow can be easily split in 4 steps:

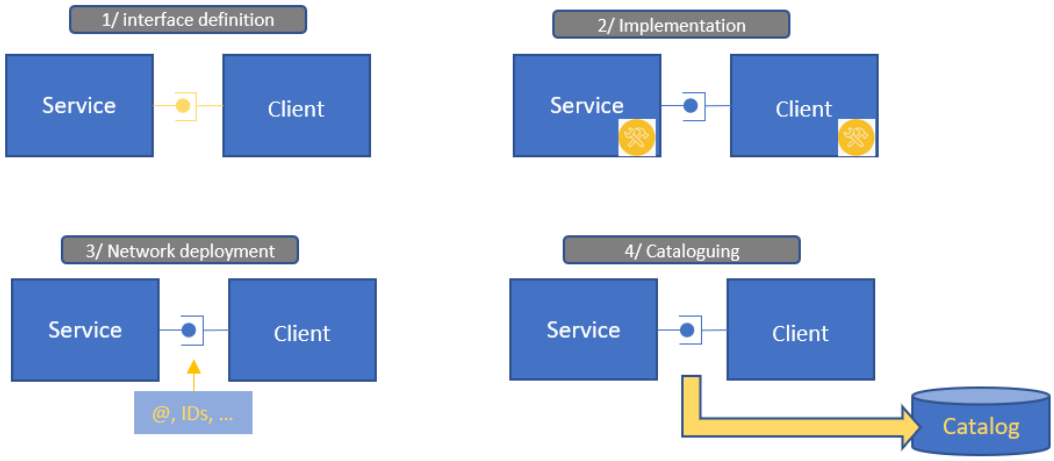

Figure 4: 4 steps of SOA introduction

- Interface definition:

This step is usually managed by the architect with an overview of the global system and the functional flow between services and clients. Architect describes the static and dynamic of the service interfaces in a model-based approach (e.g. UML).

- Implementation:

The tool chain generates skeleton code for the services and proxy code for clients. Then, developers can focus on implementing the business logic at both client and server sides.

- Network Deployment:

The network topology design and the associated configuration ruling the interconnection of services is ensured by network engineers. They will guarantee parameters consistency (IP@, service Ids, etc.) on the overall platform and the quality of service.

- Cataloguing and maintenance

Once the services are defined and integrated, it is important to maximize the reuse of existing services instead of creating a new one each time. For this, there is a need for a Change Control Board (CCB) having a central view of the overall services in the car to rationalize the usage. The CCB is responsible for challenging the need for a new service compared to reusing an existing one. The management of service versioning is also very important to avoid the multiplication of service versions along the product diversity and evolution in time. At any point of time, it shall be possible to get a clear picture of what are the service versions deployed on a product (including diversity of model) and all the clients consuming these services. If no service is consuming a version of a service anymore, we can consider removing the service to avoid unused services.

In that context, there is an open today. It is not clear whether the management of service internal to an ECU (not exposed on network) must be part of CCB process. On one hand, it is powerful for a developer to get access to all services available in the car at one glance but in another hand, the catalog may grow up very fast and it can introduce latency in the process for service definition/approval (due to CCB process) compared to a quick intra-team review of the service definition.

Note : SOA fits with legacy Model Based Design approach and it helps promoting common models across the organization. UML allows for instance to define interfaces and components implementing interfaces. We spend more time specifying each interface in detail in a model-based design approach, but at the benefit of smooth integrations with our partners and internally between different development teams.

Note : Some open source tools (e.g GENIVI YAMAICA for enterprise architect) allow to custom the UML to SOA needs by providing stereotypes (for interface definition) and convertors (UML interfaces <->Franca IDLs).

Conclusion : In a legacy organization, the CAN messages dictionary was the central point of agreement between teams dealing with inter-ECU exchanges. Developers are reluctant to define or use the message contracts at this level since it flattens the information and dilute the useful data into a flow of irrelevant parameters for developers. With SOA, each team can get their relevant data in a synthetized manner.

4.4 Security

By security in SOA, we mainly refer to mutual authentication and access control between clients and services. As a state of the art, SOMEIP standard does not bring security means.

-VSOMEIP implementation of SOMEIP standard proposes an access control level but only for intra-ECU.

-AUTOSAR ADAPTIVE brings security concepts with IAM but not compatible with COMMONAPI.

There is no comprehensive solution for SOA security as per our understanding. Then, the assets must be secured through other methods at other levels. Among the potential solutions, we can list : White listing of packet, deep packet inspection for data consistency. These security methods have the drawback to be static and introduce security elements that are in the middle of the communication path. The actual objective is to achieve a trusted authentication check both at client and server side but there is no emerging solution today that could fit a cross technology/OS SOA communication.

4.5 Semantic

In SOA, it is important the contract between parties to be unambiguous. One of the main ambiguities to remove is the format of every single data exchanged especially when it is a physical value. This often refers as the “semantic” of the data and encompass some parameters like the type, the unit, the possible range of the value (Min, Max) and if the value exchanged must be translated from implementation value the actual physical value, some translation parameters are also provided in the semantic.

Some of SOA IDL solutions come without the concept of semantic. This is the case of FRANCA IDL for instance. As FRANCA is taken as reference for RENAULT to define the interface in the car and as we want it to be compatible with the targeted technologies, these semantic parameters have been added to the IDL model. Again, there is a room for improvement on this part to ensure the compatibility between FRANCA model and AUTOSAR IDL model for instance. (note: AUTOSAR model support semantic).

4.6 SOA usage diversity in automotive

SOA paradigm of communication can be implemented in many ways. The methodology is flexible and there is a room of interpretation on some topics:

- Fine grain or coarse grain services (how complex the shared information is ?)
- Use properly the SOA communication means to ensure reliable and efficient communication.
    - Communication patterns: When to use events, fire and forget methods, methods with acknowledgement?
    - Shall we use UDP or TCP?
    - Manage the internal behavior of the service. (stateless, stateful?)

Even if there are trends on questions above (for example preferring Stateless vs. Stateful or preferring UDP vs. TCP), the decision is sometime more complex in the reality. We will elaborate in the coming sub-chapters.

4.6.1 Fine grain or coarse grain services?

To ease the decision, some strategic elements shall be considered. Automotive SOA actor shall facilitate the reuse of services across the vehicle generation or ranges (entry, mid, high tier). To achieve this, it is better to keep the services as simple as possible and prefer the multiplicity of simple services instead of few but complex. In other

words, prefer a thinner granularity of service. Nevertheless, some high-level applications cannot afford to register to dozens of services and re-implement a business logic that could be shared with other applications. For that reason, a hierarchical distribution of services is a good trade-off. This approach allows reducing the network bandwidth and MIPS consumption by introducing intermediate levels of abstraction. For example, a composed service could expose an interface to get or set the door lock status abstracting the number of doors presents in the car. The composed service would then rely on 3 or 5 others basic services controlling a single physical door. Another example could be a composed service proposing a surround view of the car by compositing the multiple video camera basic services. The composition algorithm is done once for all clients that need surround view (e.g. cluster, central panel, ADAS, …)

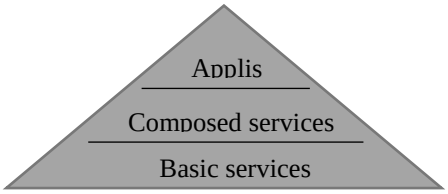

Figure 5 : services hierarchy

4.6.2 Use properly the SOA means for efficient and reliable communication

4.6.2.1 Communication pattern usage

SOA communication pattern are quite simple. It allows event-based (publish/subscribe) and method-based patterns.

Many times, the question of using one or the other is raised. Some cases are obvious like when a client wants to trigger remote function, then a "method call" will be used. For cases of information exchanged, we must question why and when the data must be consumed. Let's take an example: the central panel of the car wants to display the cellular signal strength (RSSI). Shall the modem send the RSSI periodically? on value change? is it up to central panel to request the value of RSSI in a polling mode. The 2 first rely on "event" the last on "method call".

If we consider only the efficiency of network bandwidth the event on value change seems to be the best solution but what about reliability of the information ?

Another strategic consideration is the robustness of the architecture. Compared to the legacy signal-based architecture (e.g. CAN bus), SOA brings flexibility and network efficiency but also brings some points of vigilance. Signal-based architecture can be considered "robust" against transmission errors because the signals are repeated regularly with a predefined schedule allowing the client to detect a loss of information. Client can also recover from a lost information getting the next signal.

In SOA case, this concept of repeated "signal" (a.k.a. events in SOA case) is also possible but it minimizes the SOA advantage with regard of network bandwidth. A compromise is to keep cyclic events for safety critical information and relax the cyclic constraints on other events while mitigating the risk via:

- errors monitoring at application level or

- introducing network mitigation via acknowledged transport interfaces. (for instance, TCP at OSI layer 4)

4.6.2.2 UDP vs. TCP

SOA comes with a strong adoption of UDP instead of TCP. TCP is preferred in case of large chunk of data to be transmitted [(8)] .

The reason to choose UDP are multiple: better bandwidth efficiency, lower latency, lower security attack surface than TCP stack, UDP mandatory for broadcast of data…

As soon as there is an acknowledgment at application level, the communication reliability is guaranteed even with UDP (method calls). But when we talk about broadcasting (OSI layer 3) information, there is no acknowledgement of each client for a single data. How do we know that the event has reached every registered client?

It would be possible to revert the logic and use a request/response pattern, but this is only possible for point to point communication. If we want to stick to the publish/subscribe communication logic some solutions are possible:

- Use multi-unicast using TCP instead of using broadcast with UDP. In that case, the SOA middleware will send the event to all the subscribed clients one by one and the TCP stack will guarantee the delivery at OSI level 4. This does not guarantee the delivery at application level.

- Keep UDP but have an acknowledgement of the delivery at middleware level. This is very close to the previous solution, but it is a bit more robust since it ensures there is no loss of data (due to buffer overflow for example) between the IP stack and the application (taking the assumption the middleware runs in the application execution context). Middleware knows all the subscribed clients and can wait for an ack of each middleware where the clients are hosted.

> Note: it seems that the acknowledgement of "event" was possible in previous version of SOMEIP but removed. We don't have the rational of this.

- Use the service discovery heartbeat to frequently monitor the sent events at server side and the received event at client side and compare. If not equal, then start a recovery procedure. This solution is of course only possible for services that are not time sensitive. This is conceptual only. We didn't see this solution implemented as of today.

Conclusion: for safety critical service, the events with time redundancy looks the best solution. For other services who need event-based communication, we don't see a well-defined strategy to guarantee the reliability of the service. It may be a mix of solution exposed above on a case by case basis. The criticality of the event, the bit error rate of the ethernet network, the potential high CPU load of the clients (leading to buffer overflow), could enter the picture to choose the solution.

4.6.2.3 Dynamic behavior description (stateless or stateful)

SOA paradigm is often associated to the concept of stateless of the service. If theoretically, the concept is very interesting for the flexibility and robustness it brings, in the reality it is not that easy to strictly follow the rule for car system.

For instance, if a service exposes phone capability with an SOA interface that allow to place a voice call and another to control the volume of the call, of course we cannot change the volume of the call if the call is not launched. This rule is then seen as a guideline for low complexity services but not a strict rule.

5 Outcomes

SOA adoption is getting bigger and bigger. The adaptation to Automotive constraints is still under maturation as per our understanding. Even if the basic concepts are there (IDL, transport abstraction, method and event-based communication facilities, …) , some additional concepts are not natively adopted (Security, Service discovery, …). Moreover, there is no comprehensive solution that can fit the overall diversity of ecosystems present in the car today (QNX, ANDROID, GENIVI, YOCTO, AUTOSAR, …). This induces the introduction of multiple technology bridges or compromises to make it work. This does not drastically slow-down the adoption of SOA in car industry but can create artificial silos between teams in organization (e.g. legacy vehicle teams and infotainment teams).

In another hand, we already see some benefits of SOA. In the past, we had to improve our integration camps between providers of 2 different ECUs, especially where the communication protocols, the message sets and types were not aligned prior to the developments and integration. Now using SOA principles, we can detail software contracts between ECUs and between components within an ECU in early stages of the development cycle.

Some major SOA advantages for automotive to keep in mind (not exhaustive):

- Contract based interface
- Service location agnostic

- Error detection at compilation time. (compared to thin API)
- Network bandwidth efficiency
- Code auto-generation

References :

(1):https://semver.org/

(2):https://github.com/franca/franca

(3):https://at.projects.genivi.org/wiki/display/WIK4/19th+GENIVI+AMM+Presentations?preview=/38895853/38895896/Genivi_AMM_FARA_May_2019_20190510_V2.3.pdf

(4):https://docs.google.com/spreadsheets/d/1O7gMTK1oaDHi43G2B6-Es5H4okzcjvmeRyW7sUPnAaQ/edit?ts=5bea88b1&pli=1#gid=1132768679

(5)https://fr.slideshare.net/DaiYang/scalable-serviceoriented-middleware-over-ip

(6) AUTOSAR_PRS_SOMEIPServiceDiscoveryProtocol.pdf

(7) https://docs.projects.genivi.org/vSomeIP/2.0.1/html/README.html

(8) AUTOSAR_PRS_SOMEIPProtocol.pdf (chapter 6.1)