Translation Error Handling for Multi-Protocol SOA Systems

Hasan Derhamy, Jens Eliasson
Jerker Delsing, Pablo Puñal Pereira
Luleå University of Technology
Luleå, Sweden
hasan.derhamy@ltu.se

Pal Varga
Department of Telecommunications and Media Informatics
Budapest University of Technology and Economics
Budapest, Hungary
pvarga@tmit.bme.hu

Abstract— **The IoT research area has evolved to incorporate a plethora of messaging protocol standards, both existing and new, emerging as preferred communications means. The variety of protocols and technologies enable IoT to be used in many application scenarios. However, the use of incompatible communication protocols also creates vertical silos and reduces interoperability between vendors and technology platform providers. In many applications, it is important that maximum interoperability is enabled. This can be for reasons such as efficiency, security, end-to-end communication requirements etc. In terms of error handling each protocol has its own methods, but there is a gap for bridging the errors across protocols. Centralized software bus and integrated protocol agents are used for integrating different communications protocols.**

However, the aforementioned approaches do not fit well in all Industrial IoT application scenarios. This paper therefore investigates error handling challenges for a multi-protocol SOA-based translator. A proof of concept implementation is presented based on MQTT and CoAP. Experimental results show that multi-protocol error handling is possible and furthermore a number of areas that need more investigation have been identified.

Keywords— Error handling; Protocol translation; Arrowhead; Internet of Things; Cyber-physical systems; SOA; Translation

I. INTRODUCTION

The Internet of Things (IoT) has assisted in breaking down application domain silos and promoting horizontal integration between application domains. The Arrowhead framework, presented by Blomstedt et al. in [1] is looking to improve interoperability and integrability of services provided by networked embedded devices. Cisco has estimated that there will be 50 billion devices connected to the Internet by 2020 [2]. This is a staggering number of devices and managing the differing communication standards is not trivial.

The IoT area has seen many existing and new communications protocols emerging as preferred standards. The adoption of the varied communication protocols can be linked to specific application vertical requirements and is likely to stay this way as the IoT further develops. Thus in order for the Arrowhead framework to provide interoperability between application verticals, methods and technologies for communication protocol translation are required.

978-1-4673-7929-8/15/$31.00 ©2015 IEEE

Some of the IoT protocols used in Service Oriented Architecture (SOA) based applications and systems are; Representational State Transfer (REST) over HTTP, eXtensible Messaging and Presence Protocol (XMPP), Message Queue Telemetry Transport (MQTT), Constrained Application Protocol (CoAP) and OLE for Process Control – Unified Architecture (OPC-UA). Each of these protocols offers benefits in particular application requirements, such as low-power operation, verbose headers and semantics, connection oriented messaging, decoupling producer from consumer, discovery, bootstrapping, real-time or reactiveness, and statelessness.

For the automaton domain OPC-UA is the predominant SOA protocol when communicating from the Distributed Control System (DCS) or Supervisory Control and Data Acquisition (SCADA) level and upwards in ISA-95 architecture. With the expectations of IoT devices to be used in such ISA-95 architectures, it's clear that IoT SOA protocols like CoAP, XMPP, MQTT, and REST will show up together with OPC-UA and legacy technology in large automation systems. Large EU projects like Socrates and IMC-AESOP have published several papers on such architectures and migration to such architectures [3]-[8].

Today, there is a variety of commercial IoT platforms which support interaction between different communication protocols. They offer an API for either; translation agents [9], [10] running embedded on the device or in gateways or a cloud based software bus [11], [12] for each protocol. This indicates that there is a need to integrate different communications protocols.

However, these platforms either confine applications to adapters integrated into their solutions, or require all communications be routed through a central server. Both approaches reduce flexibility for application designers and integrators, introduce security vulnerabilities with untrusted third-party clouds. This creates inefficiencies in the communications path and bandwidth usage for localized applications. Enabling protocol interoperability by the use of SOA will increase design flexibility, enable local applications and remove dependency on third-party translators. But Quality of Service (QoS), end-to-end connectivity, robustness and error handling become challenges which need to be addressed. A literature search did not reveal much research in error handling

in multi-protocol translation. This indicates the need for more research in this area.

This paper investigates the question of error handling in a multi-protocol translation for SOA systems. While in a single protocol system, errors are propagated according to protocol specification. In the case of multi-protocol systems error handling becomes more complex. In designing a SOA-based translator error handling and considerations becomes critical to robust communication. An error in one protocol must be translated to be understood by other protocols. While a SOA-based translator must also address other aspects such as QoS, control messaging, security and semantic translation, these are not considered in this paper and are considered future work.

This paper is structured as follows: Section II provides background and related work, followed by problem definition in Section III and proposed solution in Section IV. An example application scenario and implementation details and results are presented in Sections V and VI. Finally, conclusions are summarized in Section VII, with suggestions for future work presented in Section VIII.

II. Background and Related Work

A SOA-based architecture presented by Karnouskos et al. in [8] shows a shift towards SOA paradigm for Industrial IoT (IIoT). Development in the area of SOA is driven by the need for collaboration within ultra-large Scale systems [13]. SOA has been used to great effect in web based systems to create an ecosystem of collaborative parts.

The IoT is seeing growth in new application spaces and new application requirements with an evolving ecosystem of platforms, frameworks, protocols and devices [14]. This means system integrators are presented with the challenge of evolving their legacy systems, and technologies, to satisfy the new requirements and make use of new technologies. By tightly coupling translation agents into the systems the cost of upgrading the system is increased. Relying on centralized cloud based software bus also limits the ability to leverage the native benefits of using new technologies.

Collina et al. in [15] have proposed an MQTT to REST bridge. This architecture exposes MQTT topics as REST resources. This allows MQTT clients and REST clients to interact through the new centralized QEST broker.

There are two terms that appear in this field: protocol translation and protocol conversion. There is no clear differentiation between these terms, although "translation" is used generally in computer networking (especially and almost exclusively for network layer translation) - whereas "conversion" is used more widely in the industrial automation domain [16]. The traditional networking ISO-OSI terminology for nodes dealing with translation are that switches, routers and gateways work at the data link, network and transport layers, respectively [17]. The most widely known functionality is Network Address Translation (NAT), although its protocol translation version working between IPv4 and IPv6 (NAT-PT) was suggested for historic status [18] due to a series of serious issues.

There are no general guidelines or standards for higher level translation - these are seemingly all legacy solutions. Such translators do merely parameter mapping between two protocols, although sometimes also deal with the issues of the transport layers. For this study two protocols used in similar application spaces were selected. This investigation helped to refine challenges error handling and proposal of solutions to address some of these challenges. CoAP and MQTT were selected as they are both intended to be used in highly efficient industrial applications.

A. CoAP

The CoAP protocol [19] has been developed by the IETF for use in extending Internet capability down to resource constrained devices. It applies the request-response communication pattern to a client-server network model. CoAP is targeting sleepy and lossy networks in which supporting TCP becomes inefficient and power consuming [20]. It is based on UDP and provides an optional retry mechanism at the CoAP layer. It has a RESTful API with the GET, PUT, POST and DELETE verbs supported with the addition of the OBSERVE function. It creates a publisher-subscriber session between a CoAP server and client, sending notifications either when resource state changes or periodically, on expiry of 'Max-Age' [19]. Having this flexibility makes it an ideal choice for machine-to-machine interaction.

B. MQTT

The MQTT protocol has been developed for enabling efficient communication between data sources and data sinks. It applies the publisher-subscriber pattern to a client-server network model. It has recently been standardized by OASIS but has a long history with IBM being used in sensor networks. Some of its features [21] are decoupling data producer from data consumer through the centralized broker system; reduced header size and event based publishing enable highly efficient communication; QoS levels with message delivery; and, simple centralized security model with connection initiation by clients enabling useful firewall and NAT traversal features.

III. Problem Definition

Dependent on how much the protocols overlap in the OSI layers there can be much complexity in the translation process. Errors which can be detected and monitored need to be handled in a manner which will enable adequate debugging and issue resolution, either automated or by manual intervention.

In this section some of the challenges with handling errors of multi-protocol translation are elaborated and specifically, the case of translation between MQTT and CoAP studied. The error cases defined may not be exhaustive; however they represent some of the most common and in some cases challenging errors.

A. Error cases

Connection errors can occur when trying to establish new connections, having a current connection lost for some reason, or inability to close a connection gracefully. These kinds of errors need to be detected and translated appropriately to

ensure efficient use of resources (at the end points as well as at the translator). Also information about connection error events need to be made known for analysis on network performance and identifying candidates for possible improvements.

Lossy communication errors are a real problem in wireless sensor networks (WSN), which make up a good proportion of the future IIoT. The problem with lossy communication is made more complex with layers handling the issue at different layers of the OSI stack. A higher level protocol may rely on a lower level layer to guarantee transport while the target protocol may perform such transport checking itself. This means that handling lossy communication at may need to go across layers or provide informational error alert which will then rely on application layers to monitor and perform corrective action if needed.

Response related delays and application introduced delays are two such categories of delay related errors. Miss-matched timeouts at the communications or application layers can lead to one sided timeouts. To handle one sided timeouts the channels need to be re-synched, how will the translator deal with this? Application delays could be found on resource constrained devices not being able to service a request or publish an update within the time limit expected by the other party.

Application layer packetization which relies on ordered delivery by underlying layers is a special case of moving between ordered delivery protocols such as TCP and un-ordered delivery such as UDP. Is this something which can be handled by the translator? Is this something which is required by the translator?

Invalid messages arriving at the translator requires the translator to be able to detect and take appropriate action. In request-response protocols an error message can be sent to the origin. However in many publish-subscribe protocols it is not possible to send an error back. The translator must be responsible for providing some level of confidence to the end points about how much of the protocol dependencies are still valid and how much cannot be support.

Errors produced as part of the protocol procedure such as authentication problems, resource availability or others can generate legitimate error codes. These error codes while represented on one side of the translator need to be passed to the other side. To address this first the error codes of each protocol must be listed and then mapped based on the cause. Often the mapping will not be symmetric with many error codes being mapped to a lesser number, or even some error codes not being able to be mapped at all. This is described in detail in Section IV.

B. *Transient Error Cases*

Transient error cases can occur at end points but also in the translator. This class of error cases relate to errors which can occur at any time and will generally ‘heal’ in a short period of time. They require special treatment in terms of handling and translation, in terms of maintaining protocol behavior.

Transient errors due to resource usage occur when there is an increase in translation demand to such an extent that the translator is no longer capable of handling the throughput. It must take remedial action in accordance with the protocols which are being translated. These actions are highly dependent on the source of the increased demand and the capability for the translator to influence this demand. This can be illustrated where two end-points have a contract for delivery of notifications on periodic updates. If the size of the messages begins to increase while the rate of the messages remains at a high frequency, the translation of these message packets may consume resources which are not available. In this case the source of the data cannot be asked to reduce the frequency as the contract is between the end points. In such a case the translator must become a negotiating party and reduce the frequency or the size of the messages. In some cases the protocol does not allow such freedom for negotiation and in fact requires the end-point to drop the connection in such situations, as is the case for MQTT [21].

The second transient error case is from buffer problems. Buffer problems can occur when resource usage increases without correct remedial actions. But in this case we are referring to miss matches in buffer sizes between protocols and end-points. Whilst one high performance end-point, such a REST based web application, may be able to handle large verbose messages, once this is translated for a constrained end-point, such as CoAP, an error will occur between the end-points and once again remedial action will need to be taken and the event will need to be logged.

Lastly, transient errors can occur due to miss-matches in protocol or application sensitivity to jitter or delays. An application or protocol which has been designed to be sensitive to a specific range of jitter or delay may find that this is not always possible to be met either because of the jitter and delays introduced at the translator, or because the other protocol cannot guarantee the same jitter and delay bands. This can impact either the protocol behavior or the application behavior and thereby generate errors in the end-to-end process.

IV. Proposed Solution

There is much work to be done in order to address the challenges in the previous section. As each challenge is tackled there is likely to be an increase in the solution complexity. This section looks to address error code translation and discuss QoS and Error reporting.

A. *Mapping Error codes*

In this paper the error mapping was done a single pair of protocols. This is the basis for the development of an error code mapper which would produce a generic interface which individual protocols must use to define their error codes to the map. In MQTT, protocol error codes are only reported from the broker to the client [21]. The protocol does not specify error code reporting by the client to the broker. This is illustrated in Figure 1. CoAP in turn also only reports errors from the server to the client [19]. This unidirectional use of error responses means that in certain configurations the error codes cannot be passed to the end-points. This is discussed further below.

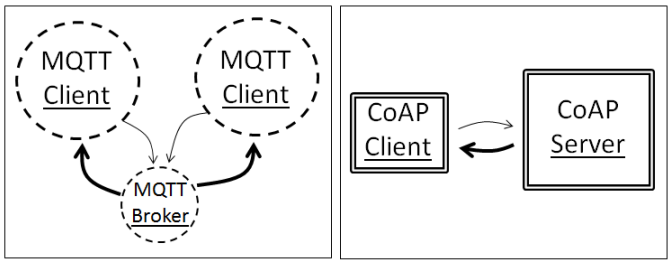

Figure 1. Direction of error reporting in MQTT and in CoAP protocols

The MQTT protocol defines error codes for initial connection and subscription control packets, other control packets do not have associated error codes [21]. The decoupled nature of MQTT networks means that clients have much less visibility of errors occurring in other clients.

Below are two tables with the error codes producible in MQTT and in CoAP. Table 1 shows the error cases which are generated by MQTT and mapped to CoAP. The mapping in this case is used when a CoAP client is attempting to initiate a subscription to an MQTT broker through the translator. This is illustrated in Figure 2.

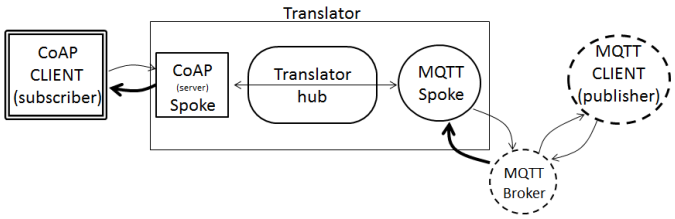

Figure 2. Configuration one allows direct translation of some error codes

In this case error codes generated by the MQTT broker can be translated and passed to the CoAP client. For example the CoAP client will be awaiting the response to its GET request and if the MQTT broker does not allow the connection or the subscription, then this error can be passed to the CoAP client.

TABLE 1. ERROR CODE MAPPING FROM MQTT BROKER TO COAP CLIENT

MQTT Error case	MQTT code	CoAP
Failure – Topic filter not accepted	SUBACK 0x80	*Error 4.04*
Not Authorized	CONNACK 0x05	*Error 4.01*
Bad username or password	CONNACK 0x04	*Error 4.01*
Server not available	CONNACK 0x03	*Error 5.03*
Identifier rejected	CONNACK 0x02	*Error 4.00*
Unacceptable protocol version	CONNACK 0x01	*Error 4.06*

In Table 2 the CoAP error codes are mapped to MQTT. These represent the case when an MQTT client subscriber is attempting to retrieve data from a CoAP server. This case is illustrated in Figure 3.

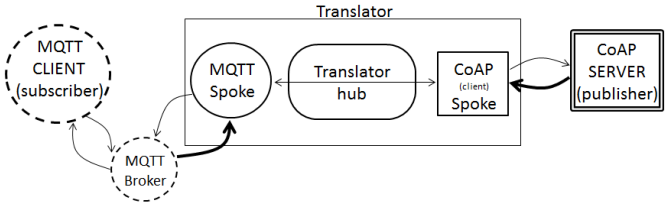

Figure 3. Configuration two does not allow direct translation of error codes

In this case there is no path for the error codes to be transferred to the subscribing MQTT client. This is due to the nature of the MQTT protocol; and so the CoAP error codes do not have a mapping to MQTT clients.

TABLE 2. ERROR CODE MAPPING FROM COAP SERVER TO MQTT CLIENT

CoAP Error case	CoAP code	MQTT
Bad Request	Error 4.00	*Not supported*
Unauthorized	Error 4.01	
Bad Option	Error 4.02	
Forbidden	Error 4.03	
Not Found	Error 4.04	
Method Not Allowed	Error 4.05	
Not Acceptable	Error 4.06	
Request Entity Too Large	Error 4.13	
Unsupported Media Type	Error 4.15	
Internal Server Error	Error 5.00	
Not Implemented	Error 5.01	
Bad Gateway	Error 5.02	
Gateway Timeout	Error 5.04	
Proxying Not Supported	Error 5.05	

However, the error codes can be used by the translator and can be mapped to translator behaviors. That is, for translating between two different protocol pairs, there will be different behavior by the translator to take corrective actions or logging. This could be translating the error code as in Table 1, or other remedial actions as defined in the translator. For the use case implemented in this paper the translator actions are described in Section VI.

B. Quality of Service aspects

Since the translator is in the path between the service producer and the service consumer application systems, its performance affects the end-to-end QoS. The translator, as a physical entity has resource limitations for memory and processing power.

Furthermore, it can serve very different application needs and very many of those. Which means it will handle many queues, which fragment those limitations further: allocating memory for the various queues, for each entity in the queue; and handling the processing overhead due to the handle queuing mechanisms (i.e. scheduling). The translator has similar types of QoS-related issues as a network-level processing node (i.e. router); although these are somewhat enlarged. This is due to the differences in information volume: translator needs to process application payload, whereas a router merely processes the network layer header.

It is not only that the translator (that handles various types of service needs) should handle QoS profiles, but these should describe further detailed metrics than those well-known at the network level. Besides handling loss, delay, delay and utilization metrics, their more specified versions [22] should be kept under control: one-way and two way throughput and delay, as well as their variance. Availability as a QoS metric is hard to address other than binary terms - either it is available, or not. Loss as a quality metric gets another meaning here - loss in translation - where not the whole message, but its parts get lost. Depending on the context and the parameters that weren't able to be mapped, may lead to QoS degradation - or it may have no noticeable effect.

C. *Error reporting aspects*

One of the most challenging aspects of distributed systems is error diagnosis. The IoT promises massively distributed systems and with the introduction of cross protocol translation the error cases not only increase in number but also in complexity. Therefore error logging and reporting is critical to the success of a translation system.

Systems can take an active approach towards error monitoring which will mean that the error notification may have soft real-time requirements. While other systems may take a reactive approach to error monitoring which will mean that they will require persistent log of the error events leading up to the final event which caused the investigation. This is true for all distributed systems with or without translation, but as the translation is a third party to the two end points involved in most interactions it becomes pertinent to discuss the implications. This means that the error stream reported by the translator must maintain a link with the end points it is translating for.

Either one of the end-points being translated or a third application will need to be able to securely and efficiently query the error log. This introduces challenges in authenticating and authorizing the interested parties to access the logs and to then find the relevant error records. If the error logs need to be exposed to a third-party which is providing support, then how can they be authorized to access the required logs?

V. Application Scenario

In order to test the proposed method in a real world monitoring application, subtask 1.8 in the Arrowhead project was chosen. Arrowhead is a European R&D project with the aim to develop SOA-based interoperable systems [1]. Arrowhead's Task 1.8 is a research and development activity aimed at delivering hardware and software for ball-bearing monitoring of a wheel loader. Task 1.8 is a joint collaborative effort conducted by Luleå University of Technology, SKF and Eistec AB. The translation scenario selected for demonstrating the challenges of error handling within the framework of Arrowhead is a CoAP based sensor network monitoring the condition of the wheel loader's ball bearings and providing this data as a service within the Arrowhead framework. See Figure 4 for a layout of the network architecture of Arrowhead Task 1.8

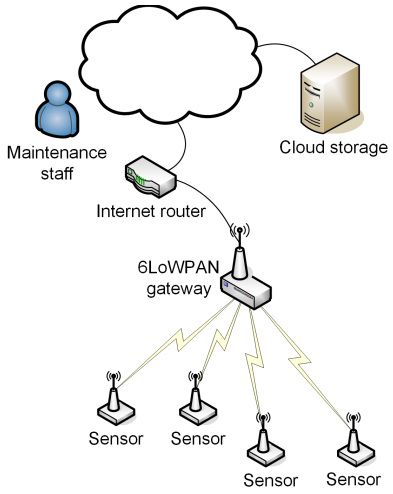

Figure 4. Wheel loader monitoring architecture

An MQTT based service consumer is behind a firewall and initiates a session with the broker consuming the sensor data from the CoAP based sensor. The MQTT service consumer could be a head office system which does not allow incoming UDP packets or a hand held device running a VPN which again does not allow incoming UDP packets. In this case CoAP is not suitable as a service consumer. A high level diagram of this scenario can be seen in Figure 5.

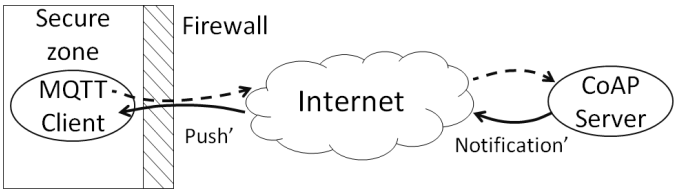

Figure 5. Headoffice analytics system; Application running in a secure area collecting data from a sensor outside the secure area.

The translator must connect the MQTT broker and the CoAP server to allow data flow. There are many error conditions possible, as stated in Section III. As a proof of concept the authors have chosen to detect a sensor disconnect at the wheel loader. In an industrial environment such as mining, road works or construction sites a disconnection error is something which the translator must be able to handle. The interaction diagram between the different components is shown in Figure 6.

Of particular interest in this scenario is the nature of CoAP running on UDP which is connectionless and therefore requires an agreed timeout based approach to connection loss. While on the MQTT side TCP is used and therefore a connection state is maintained between the end points. Even so not all TCP disconnections can be identified and so if the detection of a disconnection is desired, a heartbeat or a keep-alive timeout is also required.

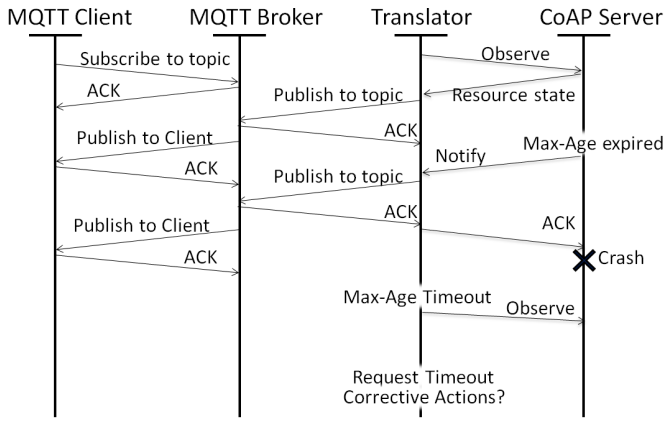

Figure 6. Error condition interaction diagram

VI. Implementation and Results

In order to validate the error case assumptions and begin the process of identifying limitations in error handling of the translation process an error translation scenario has been implemented. This section will describe the implementation setup and the results of running the error cases.

Eclipse Paho MQTT client library has been used for developing a simple visualization of data and events. This was connected to a Mosquitto MQTT broker running on a standard Windows computer on a loop back network. The wheel loader sensor was taken from the Arrowhead task 1.8 pilot project and it uses Contiki OS 2.7 and Erbium. The sensor connected to the translator through a Contiki border router and a BeagleBone Black gateway. The translator was implemented in Java and uses a Californium [23] CoAP client to initiate an observe on the state of the wheel loader sensor and using a hub and spoke architecture passes the resource notifications to an MQTT Paho client which then publishes the notification to the corresponding MQTT topic in the Mosquitto broker. This setup is shown in Figure 7 below.

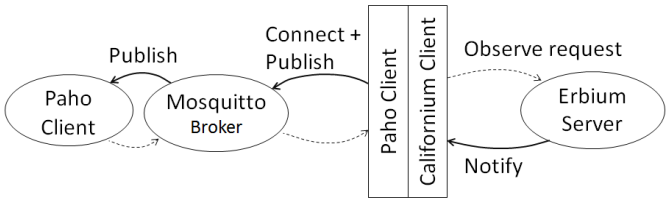

Figure 7. Implementation technologies mapped to system components

The translator itself is not the core of this paper and so the implementation has been kept to a simple transfer of payload from CoAP to MQTT. Semantics and other protocol procedures have not been considered. Using a hub and spoke architecture for implementation, results in a decoupled component based translator with only simple object method calls being setup by the hub between the spokes. This can be seen in Figure 8.

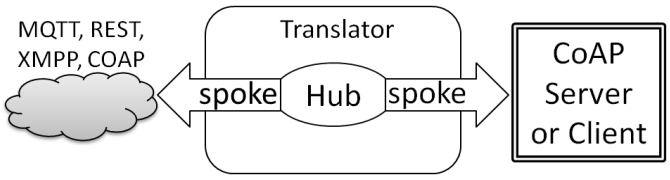

Figure 8. Translator architecture diagram

Running the experiment uncovered several error cases which were in addition to the disconnect error case that was to be modeled. This unintended error case was very useful as it shows a real world use case. The wheel loader sensor is a resource constrained device running on a low power network and therefore notification timeout due to late delivery or packet loss was common. In these cases the CoAP spoke would follow the CoAP specification and on max-age expiry, would attempt to re-register the observation. This was in almost all cases successful and would re-establish the periodic notifications. However in the MQTT specification there is no mechanism within the protocol to pass information regarding update timeout except by disconnection. The keep-alive timer was controlled by the Paho library and so would keep the connection alive even when no data was being sent. This means that a non-standard message would need to be sent from the CoAP spoke to inform the MQTT client that the sensor has had a timeout. Processing of this message would be at the discretion of the MQTT client application.

However, in the event of a disconnection error which does have protocol procedures in both MQTT and CoAP there is still special behavior required. So the CoAP spoke monitors the max-age of the last resource state update. If this max-age is exceeded then a timeout is noted and the CoAP spoke will attempt to re-establish the observation, as described earlier. However if the re-establishment is not successful then the CoAP spoke cancels the observation and an error passes an error event to the translator hub. In a normal setup an ungraceful disconnection detected by the MQTT system, would result all subscribing clients being delivered a last will message, if it is available. However in this case the MQTT system does not have visibility of the CoAP disconnection. It is required to translate and notify the MQTT system of this disconnection event. The proposed translator maps the CoAP disconnection to a last will message in the MQTT side. This mapping takes place in the translation hub. Figure 9 shows the interaction between internal components of the translator system.

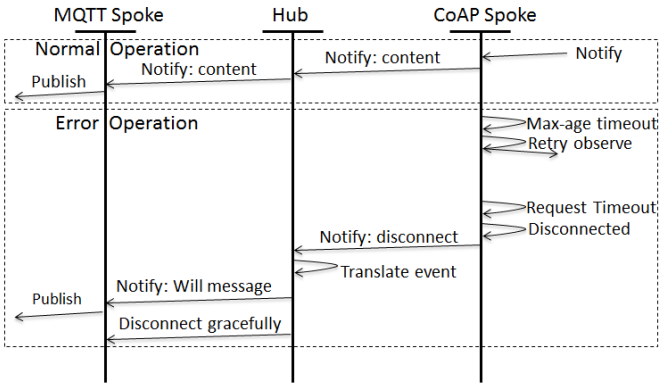

Figure 9. Internal object interaction of the translator behavior

In this way, the translator has made use of the protocol procedures of both sides to make sure both protocols are aware of the error event. For interoperable use of the translator between systems the definition of both the timeout event and the disconnect event signals is a must. For this implementation two event signals were defined and encoded in XML. These tags are presented below.

```
<s n="event" sv="disconnect"/>
<s n="event" sv="timeout"/>
```

Implementation of the rudimentary good path payload translation between CoAP and MQTT was relatively trivial. However implementing the mapping of disconnection and timeout errors introduced a lot of complexity to the code. By refining the implementation the error handling and mapping were moved to the central hub. This reduced the complexity of the translation effort immensely. Each translation spoke did not need to have knowledge of the other. This means that once a spoke is developed it can be connected through the hub to any other spoke. This reduces the effort required to develop translation services between the ever changing array of protocols.

By decoupling the protocol specific handling to spokes and translation aspects to the translation hub has meant that the solution is extensible for new spokes and allows interesting potential for a multi-spoke translator. It has also meant that error cases can be handled in a standard manner within the translation hub and new spoke development need only use available hooks in the translation hub in order to pass error conditions.

The results were promising with key advantages to the use of a hub and spoke SOA based translator. Its active participation in the network, its simplicity for handling errors and potential for extension to being orchestrated and also into semantic translation are the main advantages.

VII. Conclusion

This paper has presented the challenges and solution for error handling in multi-protocol translation scenarios for SOA systems. This work is motivated by the creation of new systems-of-systems that are composed of application domains with different communications requirements. Current protocol translation solutions use tightly-coupled software components or integrated middleware that reduces flexibility and increases cost of change. Moreover, utilizing centralized software bus for translation increases round trip time, bandwidth usage and introduces further dependencies (i.e. on cloud platforms, often operated by third parties). In both of these cases intermediary protocols are used and this limits the benefits of the native communications protocol.

On the other hand, SOA-based translation systems provide the opportunity to decouple the translation components from the application development and also create flexibility in deciding execution location of the translation service.

This paper discussed the challenges of error handling in the case of loosely coupled SOA translators. Some of the investigated error cases are connection errors, lossy communication, application introduced delays and protocol error code mapping. Beside these, the transient errors in the translator and at end-points need to be handled. This means that not just message parameter mapping, but the protocol procedures of one side needs to be reflected on the other side. Transient errors can occur, when resource requirements in terms of memory and processing power do not scale as usage demand increases, a mismatch in buffer requirements between a protocol pair, or a mismatch in jitter and delay sensitivity. These transient errors require the translator to be capable of self-monitoring and also negotiation capabilities with the protocol pairs.

The proposed solution uses translator behaviors, which are then mapped to a protocol procedure or error code. The translator's overall behavior depends on the protocols being translated. This is realized in the implementation by the use of a hub and spoke architecture with the hub containing the possible behaviors of each spoke.

The proof of concept implementation provided error handling for translation between CoAP and MQTT. To accomplish this, it was required to pass error cases generated on the CoAP side of the connection to be communicated to the MQTT side. It was decided to define a set of xml tags and attributes, which would communicate the errors at the application level. There were two such events defined for this proof of concept, they were the disconnect event and the timeout event.

VIII. Future Work

In the future, the architecture of the multi-protocol translator needs to be defined and refined. Use case extension to other SOA protocols such as XMPP and REST will also be needed.

Challenges are seen in orchestration and co-ordination of the translator end-points, managing resource requirements, providing security in terms of privacy, confidentiality and authenticity, and proving performance and flexibility gains.

Performance metrics, evaluation and bench-marking will be needed in order to prove the advantages of a multi-protocol SOA translator. Further development of the semantics used to send error information and signals should be looked into.

Error logging and diagnosis has much work to be done. Logging encompasses a larger scope than just error events should enable SOA based applications to create an end to end stream of events. An API must be developed with the ability for machine query and manual query of the logs.

ACKNOWLEDGMENT

The authors would like to express their gratitude towards the European Commission and Artemis for funding, and our partners within the Arrowhead project.

REFERENCES

[1] Blomstedt, F.; Ferreira, L.L.; Klisics, M.; Chrysoulas, C.; Martinez de Soria, I.; Morin, B.; Zabasta, A.; Eliasson, J.; Johansson, M.; Varga, P., "The arrowhead approach for SOA application development and documentation," *Industrial Electronics Society, IECON 2014 - 40th Annual Conference of the IEEE* , vol., no., pp.2631,2637, Oct. 29 2014-Nov. 1 2014. doi: 10.1109/IECON.2014.7048877

[2] D. Evans, "*The internet of things how the next evolution of the internet is changing everything*", White Paper, Cisco, San Jose, CA, April 2011.

[3] Colombo, A. W.; Bangemann, T.; Karnouskos, S.; Delsing, J.; Stluka, P.; Harrison, R.; Jammes, F.; Lastra, J. L. M., *Industrial Cloud-Based Cyber-Physical Systems - The IMC-AESOP Approach*. Springer, 2013.

[4] Delsing, J.; Carlsson, O.; Arrigucci, F.; T. Bangemann, T.; Hübner, C.; Colombo, A. W.; Nappey, P.; Bony, B.; Karnouskos, S.; Nessaether, J., *"Migration of scada/dcs systems to the soa Cloud,"* in *Industrial Cloud-Based Cyber-Physical Systems.* Springer International Publishing, 2014, pp. 111_135.

[5] Jammes, F.; Bony, B.; Nappey, P.; Colombo, A. W.; Delsing, J.; Eliasson, J.; Kyusakov, R.; Karnouskos, S.; Stluka, P.; Tilly, M., "Technologies for soa-based distributed large scale process monitoring and control systems," in *IEEE IECON 2012*. IEEE, 2012.

[6] Jammes, F.; Karnouskos, S.; Bony, B.; Nappey, P.; Colombo, A. W.; Delsing, J.; Eliasson, J.; Kyusakov, R.; Stluka, P.; Tilly, M., "Promising technologies for soa-based industrial automation systems," in *Industrial Cloud-Based Cyber-Physical Systems*. Springer International Publishing, 2014, pp. 89_109.

[7] Karnouskos, S.; Colombo, A. W.; Bangemann, T.; Manninen, K.; Camp, R.; Tilly, M.; Sikora, M.; Jammes, F.; Delsing, J.; Eliasson, J., "The imc-aesop architecture for cloud-based industrial cyber-physical systems," in *Industrial Cloud-Based Cyber-Physical Systems*. Springer International Publishing, 2014, pp. 49_88.

[8] Karnouskos, S.; Colombo, A.W.; Bangemann, T.; Manninen, K.; Camp, R.; Tilly, M.; Stluka, P.; Jammes, F.; Delsing, J.; Eliasson, J., "A SOA-based architecture for empowering future collaborative cloud-based industrial automation," *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society* , vol., no., pp.5766,5772, 25-28 Oct. 2012 doi: 10.1109/IECON.2012.6389042

[9] Cumulocity. (2015, April 14). *Interfacing devices* [Online]. Available: http://www.cumulocity.com/guides/concepts/interfacing-devices/

[10] IoTivity. (2015, April 14). *Protocol plug-in manager* [Online]. Available: https://www.iotivity.org/documentation/iotivity-services/protocol-plug-manager/

[11] PTC. (2015, April 14). *Thingworx platform a ptc business* [Online]. Available: http://www.thingworx.com/

[12] Xively. (2015, April 14). *Xively rest api* [Online]. Available: https://personal.xively.com/dev/docs/api/

[13] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, and K. Wallnau, "Ultra-Large-Scale Systems - the software challenge of the future," Software Engineering Institute, Carnegie Mellon, Tech. Rep., Jun. 2006. [Online]. Available: http://www.sei.cmu.edu/library/assets/ULS_Book20062.pdf

[14] Suresh, P.; Daniel, J.V.; Parthasarathy, V.; Aswathy, R.H., "A state of the art review on the Internet of Things (IoT) history, technology and fields of deployment," *Science Engineering and Management Research (ICSEMR), 2014 International Conference on* , vol., no., pp.1,8, 27-29 Nov. 2014 doi: 10.1109/ICSEMR.2014.7043637

[15] Collina, M.; Corazza, G.E.; Vanelli-Coralli, A., "Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST," *Personal Indoor and Mobile Radio Communications (PIMRC), 2012 IEEE 23rd International Symposium on* , vol., no., pp.36,41, 9-12 Sept. 2012. doi: 10.1109/PIMRC.2012.63628

[16] Zhang, H.; Lou, G.; Wang, H.; Sun, Z., *"Development for Protocol Conversion Gateway of Industrial Field Bus"*, in *Advanced Technology in Teaching - Proceedings of the 2009 3rd International Conference on Teaching and Computational Science*, Y.W. Wu, Editor. 2012, Springer-Verlag Berlin: Berlin. p. 211-216.

[17] B. Singh, "*Data Communications and Computer Networks*", 3rd ed. New Delhi, India. PHI Learning Pvt., 2011.

[18] C. Aoun, E. Davies, "Reasons to Move the Network Address Translator - Protocol Translator", *IETF RFC-4966*, 2007.

[19] *The Constrained Application Protocol (CoAP)*, IETF RFC 7252, 2014 [Online]. Available: https://tools.ietf.org/html/rfc7252

[20] Bormann, C.; Castellani, A.P.; Shelby, Z., "CoAP: An Application Protocol for Billions of Tiny Internet Nodes," *Internet Computing, IEEE*, vol.16, no.2, pp.62,67, March-April 2012. doi: 10.1109/MIC.2012.29

[21] *MQTT Version 3.1.1. 2014*. OASIS Standard [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

[22] Varga, P.; Moldovan, l., "Integration of service-level monitoring with fault management for end-to-end multi-provider ethernet services," *Network and Service Management, IEEE Transactions on* , vol.4, no.1, pp.28,38, June 2007 doi: 10.1109/TNSM.2007.030103

[23] Kovatsch, M.; Mayer, S.; Ostermaier, B., "Moving Application Logic from the Firmware to the Cloud: Towards the Thin Server Architecture for the Internet of Things," *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on* , vol., no., pp.751,756, 4-6 July 2012 doi: 10.1109/IMIS.2012.104