Comparative Influence Evaluation of Middleware Features on Choreography DSL

Nebojša Taušan, Pasi Kuvaja, Jouni Markkula, Markku Oivo
Department of Information Processing Sciences
University of Oulu
Oulu, Finland
{nebojsa.tausan, pasi.kuvaja, jouni.markkula, markku.oivo}@oulu.fi

Jari Lehto
Department for Process Improvement
Nokia Siemens Networks
Espoo, Finland
jari.lehto@nsn.com

***Abstract*—Domain-Specific Languages for service interaction modeling in the embedded systems domain are generally considered insufficiently expressive. To fully represent what is relevant for the developers, service interactions are commonly modeled from two viewpoints: orchestration, which is the individual, and choreography, which is the global viewpoint. In the embedded systems domain, proposed modeling languages are focused on orchestrations, while choreography modeling is neglected. For this reason, we compared two middleware products, one from the automotive and the other from the telecom industry sector, and analyzed variations in the implementation of choreography relevant features. Our analysis shows the influences of implementation variations on language for choreography modeling. Our findings can be useful in developing a domain-specific language that will allow the full representation of choreographies in the embedded systems domain.**

Keywords-choreography; DSL;middleware; SOA; MDE

I. Introduction

Service-Oriented Architecture (SOA) is an architectural style that is commonly used in the development of large enterprise systems [1]. Recently, SOA has found its application in industrial sectors such as the automotive and telecom where it is used in the development of embedded systems [2] [3] [4]. This has opened an opportunity to transfer knowledge and technology from one domain to another, but also to extend existing knowledge and technology, so it can meet new challenges that are specific to the embedded systems domain.

Systems built based on the SOA style can be described as collections of autonomous applications, called services, which interact to fulfill the stakeholder's needs. Therefore, explicit representation of how services interact becomes an important aspect of SOA systems. According to Dijkman and Dumas [5] and Peltz [6], modeling of the service interaction aspect should comprise two viewpoints, orchestration and choreography. In short, orchestration shows service interactions from a single participant's point of view, while choreography shows a global, peer-to-peer interaction between participants. These viewpoints overlap in the sense that both illustrate how underlying services interact, but differ in the perspective, or in the viewpoint, from which they show the interaction aspect.

One approach to how service interaction aspect can be analyzed and specified is to use Domain-Specific Languages (DSL). DSLs, unlike general purpose languages are focused on one particular aspect or one particular domain of a software system. The main idea behind DSL usage is to shorten the development time, reduce errors, and improve the communication by enabling language support for concepts that are specific to the aspect of interest [7].

Modeling of service interaction aspects in the enterprise system domain is supported with several DSLs; examples are [8] [9] [10] [11] [12] for orchestration and [13] [14] [15] [16] for choreography. These languages, however, are not sufficiently expressive to represent interactions that may occur in embedded systems [17] [18] [19]. Therefore, new DSLs, or supplements to existing DSLs, have been developed for embedded systems.

In the telecom domain, Call Control eXtensible Markup Language (CCXML) [20] is used to controls the invocation order of telephony services. The drawback of CCXML is that it can invoke only services developed in telephony specific technologies. To overcome this limitation, a State Chart eXtensible Markup Language (SCXML) [21] was proposed. SCXML is a generic language for describing complex state machines. It complements CCXML by providing a generic state-machine framework and by enabling it to invoke services developed in telephony and non-telephony-specific technologies.

Vandikas and Niemoeller proposed SCALE [22] as a modeling language whose main goal is to enable modeling of telecom specific interactions, but also to allow convergence of telecom services and services developed in different domains and technologies. Similarly, SPATEL [23] language targets the same problem, and offers technology-independent primitives that can be used for the development of telecom services where large numbers of resources needs to interact over different protocols.

The described DSLs enable service interaction modeling; however, they support only the modeling of individual participant point of view, or orchestration. Global view on interactions, or choreography, is not natively supported with their language entities.

Service interactions can be modeled with domain-agnostic languages or by modifying languages from different domains. A case in the automotive domain is reported by Fiadeiro et al. [24] where SENSORIA

Reference Modeling Language (SRML) is used. SRML [25] is designed to be a domain-agnostic language, with strong expressiveness for SOA, and to be easy for formalization.

Business Process Execution Language (BPEL), which is used for modeling enterprise service interactions, is modified by Iwai et al. [18] to represent the complex interactions of services in automotive domain. As in the case of telecom DSLs, these approaches target only the orchestration point of view. With the exception of the work by Tsai et al. [26], the current state-of-the-art in service interaction modeling led us to conclude that less focus is put on modeling choreography aspects in service-oriented embedded systems development.

To bridge this gap, part of the work done during the AMALTHEA [27] research project was to develop a DSL suitable for choreography modeling in the embedded systems domain. The language is one of the several tools in the tool-chain platform that is implemented within the project. During implementation tasks, we adopted the guidance for DSL development proposed by Merink et al. [28]. This guidance is organized according to DSL development phases, and in this article, we will present our findings from the analysis phase. During this phase, together with industry partners, we analyzed middleware products that are used in the automotive and telecom industry. There are two main reasons why middleware analysis is relevant for the development of DSL.

The first reason originates from the DSL development guidance [28], according to which the input to the DSL analysis phase can be technical documentation, knowledge provided by experts, customer surveys, and the existing source code base. Accordingly, for our analysis, we used expert's knowledge, and technical documentation of two middleware products. Middleware, and its documentation, is an unavoidable part of any large software system, and its main responsibility is to enable seamless interaction between system parts [29]. Accordingly, it is a valuable source of service interaction-related knowledge, which is the key result of the DSL analysis phase.

The second reason is related to Model-Driven Engineering (MDE) [30], which is the engineering approach in companies that participated in this project. In the MDE approach, relevant system aspects are modeled using DSLs. Unlike in traditional, document-driven approaches, the developed models in MDE are executable or readable by tools. This allows automatic analysis, transformation from one system representation (one model) to another, and automatic test and source code generation. Source code generated from different models relies on, or executes on top of, middleware. Middleware, however, imposes rules and constraints to that code that must be understood and followed during modeling [31]. One way to enable this is to include and enforce those rules and constraints with DSLs. This way, DSLs and their models become tightly coupled to the middleware on top of which the developed application will execute.

Middleware products support developers by providing them with features that hide complex low-level tasks [29]. Different middleware products, however, implement features differently, which introduces variations in implementation and in extent of support the feature provides. If features, with the rules and constraints they impose, are to be addressed with DSL, these variations must be taken into account. To better understand the relationship between variations and DSL development, in this study, we will answer the following research question.

How do variations in the implementation of middleware features influence the implementation of the DSL for choreography modeling?

Answering this research question will help choreography DSL developers by pointing out which language entities are influenced by feature implementation variation and how. To answer this research question, we identified choreography-relevant features and their implementation variants (Section II). Based on these features, we compared two middleware products, identified influenced choreography language entities, and described the influence in more detail (Section III). Following is the discussion on benefits that can be expected from DSL that includes implementation variations (Section IV). Finally, we summarize the study findings and describe the future work (Section V).

II. Research Design

Analysis phase of DSL development is conducted by adopting DESMET [32] approach for evaluation, and Goal Question Metrics (GQM) method [33] for feature and scale derivation. DESMET proposes nine methodological approaches for evaluating methods, tools, and technologies [34], and defines the criteria based on which an evaluator can select the most appropriate one. Based on the evaluation context, nature of the impact, nature of the evaluation object, and maturity of the item criteria, we have selected *feature analysis in screening mode* (FA) approach for this study. The evaluation context criterion recommends FA in cases where the object under evaluation will be sold as a part of a larger product. Middleware, as the object under evaluation, is a part of the overall system that resides between the operating system and application. The nature of the impact criterion recommends FA in cases when a study produces qualitative results. This is in line with this study, since we are aiming to show the influence of implementation variations on DSL development. The nature of evaluation object criterion recommends FA when tools are in the focus of evaluation. Middleware is primarily a technology, but it can also be approached as a tool for supporting a developer's work. The maturity of the item criterion proposes FA when large amounts of information about study object are available. This corresponds with the middleware products evaluated in this study. The first is the de facto standard in the automotive industry. The second is a proprietary technology owned by the company that participates in this research project.

A. Analysis Procedure

DESMET FA is a qualitative approach to evaluation. It formulates features according to what users expect from the method, tool, or technology, and derives corresponding scales that measure the extent to which the candidate method, tool, or technology conforms to the formulated features. When FA is done in the screening mode, feature derivation and evaluation is done by a single person based on public documentation only. Accordingly, during this study, middleware features that are seen as relevant for choreography DSL are identified, their scales are derived, and, based on those, two middleware products are evaluated. Contrary, instead of one, four researchers and one industry expert collaborated during feature derivation and evaluation. Research collaboration consisted of face-to-face meetings, teleconference meetings, and exchange of email messages.

DESMET FA evaluation consists of six steps, which we followed during this study, and described in the text below.

Step 1: Identify the candidate method/tool/technology. This research project, brought together researchers and experts from automotive and telecom industry. In both industries, different middleware products are used for systems development and for this analysis, AUTOSAR [35] and LISA were chosen. The reason for choosing AUTOSAR, over other products such as OSGi, is that it represents a de facto standard in automotive industry. It is a result of a global partnership of automotive manufacturers and suppliers, which aims to become the standardized architecture for automotive software. AUTOSAR is also a dominant middleware in automotive companies which participated in this research project. LISA stands for Light Intelligent Software Architecture and it is a proprietary middleware solution for the development of telecom systems. The reason for choosing LISA for this analysis is that it is still a prototype and open for modifications. This motivated telecom experts to compare LISA against more mature AUTOSAR and to learn about similarities and differences in the implementation of two products.

Regardless of differences in many aspects of automotive and telecom systems, closer inspection of the middleware products revealed a number of similarities. These similarities form a basis for comparing AUTOSAR and LISA. Figure 1, illustrates the similarities between the two systems, and shows the position of middleware within them. With reference to Figure 1, these similarities are: a) Systems consist of heterogeneous hardware devices (Hardware A, B, and C). b) Hardware devices are interconnected with heterogeneous network technologies (labeled with 1 and 2). c) Hardware devices can have different operating systems (OS 1, 2, and 3). d) The middleware homogenizes hardware devices, network and operating systems. e) The middleware hides hardware, network, and OS complexities by offering higher level application programing interface (API) to application components. f) Application components (C1–5) reside in hardware devices and run on top of middleware. g) Applications are realized with one or more application components. h) Hardware and application components may or may not be under the control of a single authority (Hardware A and B belong to D1 domain, Hardware C belongs to D2 domain, while domain here denotes different organization units or different companies). i) End-user perceived functionality (Functionality 1 and 2) is realized through application component interactions. j) Application components that realize functionalities can reside on the same or on different hardware devices. Described similarities are the key argument why we consider AUTOSAR and LISA comparable and therefore they will be explained in more detail.

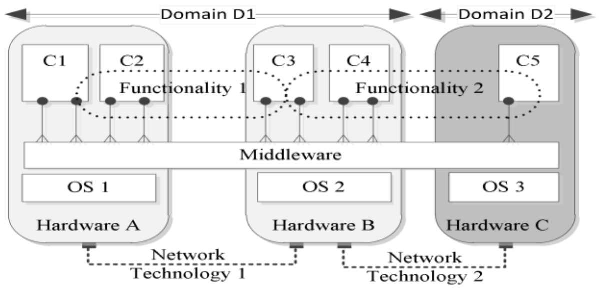

Figure 1. Architectural similarities of two systems

Step 2: Devise the assessment criteria. FA is a comprehensive approach to evaluation. Besides technical issues, the method proposes to evaluate features from economic, cultural, and different quality aspects such as maintainability or portability. To narrow down the scope of evaluation, we applied the GQM method during the derivation of features and corresponding judgment scales. The importance of the clear goal definition is highly stressed in the GQM approach since it provides a converging point for future scales and it reduces the number of possible measurements [36]. It is important to note the misalignment in terminology within DESMET and GQM. In DESMET, judgment scales are used to estimate the derived features, while in GQM, scales are used to estimate the measurements. Therefore, in this study, the terms measurements and features can be considered equivalent since both are used for answering questions formulated according to a specified goal. Goal specification is further facilitated with a GQM template [33], which consists of five key-value tuples. A study goal, based on this template, is presented in Table I.

The first tuple in the template defines the object under investigation. In this study, the object under investigation is the middleware. The second tuple defines the purpose for analyzing that object. In this article, the purpose is to learn which middleware features should be considered during the development of choreography language. Accordingly, the choreography aspects of service interaction support in middleware are the quality focus against which we analyzed LISA and AUTOSAR. Viewpoint narrows the scope of learning by focusing it on a specific role in the development process. We selected the software architect role because it is responsible for middleware-related decisions, and because LISA and AUTOSAR can be easily compared in architectural terms such as components, services, interface, and message. Lastly, this international research project is

TABLE I. FEATURE ANALYSIS GOAL BASED ON GQM TEMPLATE

Key	Value
Analyze the :	Middleware
For the purpose of:	Learning
With respect to (quality focus) :	Service interaction aspects relevant for choreography modeling
From the viewpoint of:	Software architect
In the context of:	Research project

the context in which the evaluation took place.

Specified in this way, the goal guided our collaboration with the industry experts and our study of the literature. This resulted with the definition of three questions whose answers will contribute to the goal accomplishment. Questions are broken into features relevant for the choreography DSL, and for each feature, a corresponding ordinal scale is derived. Here, we will emphasize that scales are derived based on the extent of support the feature provides to developers. A higher feature score corresponds to higher flexibility, less effort, and less cognitive burden for developers. Scales do not measure the variations in technology that is used for feature implementation.

The first question, based on the defined goal, is: How does middleware support the invocation of services offered by different systems or system parts? To answer this, three features are identified and explained in the following text. *Functionality access* is the first. It concerns middleware support for invoking services that use different interfacing technologies, e.g., Web Service Description Language (WSDL) and Interface Definition Language (IDL). *Location transparency* is the second. It concerns middleware support for binding service requesters and providers. Location Transparency can be realized using requester's criteria based on which middleware selects the provider, logical names based on which physical location of the provider is resolved, or by plain routing. *State information* is third. It concerns types of state information that middleware monitors. State information types are classified into service, session, and functions categories. Service indicates the state of the application or component that implements the service. Session indicates the state of the interaction between two services. Function indicates the state of the composition of services that fulfills a system-level task. Table II shows the extent of support the middleware provides for identified features.

The second question based on the defined goal is: How does a middleware product supports issues related to messages? Message is used in a broad meaning, and covers both the format of the message and the type of data that is carried. To answer this question, three features are identified and explained. First is *message format*. It indicates which message and data formats can be processed. Message format examples can be Session Initiation Protocol (SIP) or Simple Object Access Protocol (SOAP), while the data format can range from streams of bits to documents written in plain text or in eXtensible Markup Language (XML). Data format is commonly defined by the message format that carries it. Second is *message transformation*. It concerns middleware support for transformation of messages from one format to another. Third is *interaction scenario*. It shows the middleware ability for processing predefined ordering of message exchange occurrences. These features and extent of support are given in Table III.

TABLE II. FEATURES FOR SERVICE INVOCATION SUPPORT

Feature	Scale	Scale description (implementation variants)
Functionality access	2	Middleware supports standardized interfacing technology specific for an industry sector
	1	Middleware supports key interface technologies
	0	Middleware imposes single interface technology
Location transparency	2	Middleware selects service provider, resolves its location and routs the request
	1	Middleware resolves service provider's location and routes the request
	0	Request contains details that are necessary for binding (provider name, physical location, etc.). Middleware only routs request to provider
State information	2	Middleware provides state information on function, session, and service level
	1	Middleware provides state information on session, and service level
	0	Middleware provides state information on service level

TABLE III. FEATURES FOR MESSAGING SUPPORT

Feature	Scale	Scale description (implementation variants)
Message format	2	Middleware processes message format that is standardized within an industry sector
	1	Middleware processes key message formats that are used in an industry sector
	0	Middleware imposes message format
Message transformation	2	Middleware transforms key message formats, and allows developers to create custom pluggable transformation additions
	1	Middleware transforms key message formats; middleware vendors supply additional transformations through product updates.
	0	Middleware does not provide message transformation services
Interaction scenario	2	Middleware processes custom definitions of message interaction scenarios
	1	Middleware service supports generic interaction scenarios
	0	Middleware does not support the processing of interaction scenarios

Lastly, a third question based on the defined goal is: How is a message transmitted from its origin to its destination? Two features are identified and explained. *Protocol support* is the first, and it shows middleware support for a variety of communication protocols. As is the case with messages, the term protocol here is used to cover all types of protocols, ranging from lower-level network specific protocols, such as Controller Area Network (CAN), to high level application protocols, such as Hypertext Transfer Protocol (HTTP). *Protocol translation* is the second identified feature, and it shows how middleware supports the translation of one protocol to another. These features and implementation variants are shown in Table IV.

Step 3: Compiling information about the study object. To evaluate the candidate technologies, relevant

TABLE IV. FEATURES FOR MESSAGE TRANSMISSION SUPPORT

Feature	Scale	Scale description (implementation variants)
Protocol support	2	Middleware supports different protocols by providing protocol-independent communication service
	1	Middleware supports different protocols by providing protocol-dependent services
	0	Middleware imposes the protocol
Protocol translation	2	Middleware communication services hide protocol translations from services
	1	Middleware provides distinct services for protocol translation
	0	Middleware does not provide translation support. Services are responsible for translation

documentation needs to be collected and studied. This research project provided a context that allowed us to collect high-quality, company-specific documents, and to capture the knowledge of company experts in meeting notes, email discussions, and workshop summaries.

Step 4: Scoring of features. Based on the gathered information, middleware products are evaluated against derived features. The process of scoring consisted of the initial score proposals and discussion. During score proposal, each team member proposed a score for each feature. During the discussion, the differences in score proposals are aligned.

Step 5: Analysis of the score. To decide which method, tool, or technology best fits the needs of the most target users, feature scores are analyzed. The goal of this evaluation, however, is not to select between middleware products. Our goal is to learn about middleware features and to show how their variants can have an influence on the language for choreography modeling. For this purpose, we used the meta-model for choreography language defined in [37]. This model defines what is necessary for the development of global interactions, and represents a foundation for the development of choreography modeling languages. It consists of attributes enclosed in entities that are interconnected and grouped into model subsets. We used this model to identify how, and which of its entities are influenced by the variations in middleware's feature implementation.

Step 6: Presenting a report on the evaluation. The research findings are summarized in a technical report.

III. Research Findings

The research findings are divided into two groups. The first group consists of feature scores and the rationale behind scoring. In the second, we explain how variations in feature implementation have an influence on DSL for choreography modeling.

A. Features Scores

The rationale behind scoring is based on an in-depth analysis of the technological solutions and concepts that are used for feature implementation and on industry expert's evaluation of the extent of support the feature provides. The implementation details used in AUTOSAR and LISA for an identified feature are described below.

Functionality access: To describe what a service can provide, what other services it uses, and how to invoke the service, AUTOSAR developed the AUTOSAR Interface [38]. This interface has a formal structure that describes all aspects required for the invocation of functionality. LISA, in contrast, has no structured description of a service. LISA facilitates access to service functionalities by offering a proprietary API through which applications (services) register and publish their functionalities. Through this, potential clients are able to invoke the functionalities they need. Other than function names, no additional details are provided.

Location transparency: Both middleware products studied in this evaluation provide support for binding by hiding the location details of services. A service can invoke another services' functionality using only its logical names, while the middleware pairs logical names with the services' functionality and its physical location. This allows services to be moved to different hardware devices, and if there is a need, to change its implementation details. Since the functionality is invoked using logical names, flow of service interactions is not affected.

State information: Both AUTOSAR and LISA provide state information on the service and session levels. On the service level, AUTOSAR monitors the state of the runnable concept [38], while LISA allows for monitoring of each service that implements the proprietary LISA-specific addresses. On the session level, AUTOSAR's inter-runnable communication state information is provided with global variables and/or shared memory monitoring [38], while LISA provides session-level information by monitoring its implementation of message queues.

Message format: AUTOSAR services exchange information using three standardized variable groups: data element, mode declaration, and application error [38]. The data element is the piece of information transmitted between services. This information is sent to, and received from, the service's operations, and it can be any primitive type, such as integer or float, or a collection of primitive types referred to as the complex type. Mode declarations define data for the service mode configuration, while application errors carry the information about error occurrences within a service or during communication. In AUTOSAR, variables are exchanged by passing them to functions directly, and no additional messaging technology is used.

Messages in LISA are exchanged using proprietary messaging technology. Before a message of any type is sent, it is wrapped up in a LISA-specific message format and routed to the destination. On arriving at the destination, it is unwrapped and parsed by the receiver.

Message transformation: In AUTOSAR, the object of transformation is the data element variable group, and this task is appointed to the runtime environment (RTE). Transformation definitions are provided by developers, and, based on them, RTE can perform several types of transformations. Examples are transformations to/from different linear-scaled data representations, different text-table data representations, and transformation of composite data representations [39]. LISA does not provide any features for transforming messages. Instead, it is the responsibility of the sending or receiving service to preprocess the message so that it can be used by the receiving application.

Interaction scenarios: There are two generic scenarios that describe a message exchange in AUTOSAR, Client-Server and Sender-Receiver [39]. Client-server involves the client, who requires the functionality and server that provides that functionality. The client initiates the communication by requesting the server to perform the functionality and if necessary it provides one or more parameters. The server performs the required function, and dispatches a response to the client. Invoking a function is performed by RTE, and these invocations can be either asynchronous or synchronous. The sender-receiver involves the sender of the message and one or more receivers. This is one way, the asynchronous interaction scenario, and any reply sent by receiver is seen as a separate sender–receiver communication. The same scenarios exist in LISA. The difference is that in the case of AUTOSAR these patterns are explicitly defined within the interface, while in LISA, no such definition exists. The client-server scenario occurs when one service invokes the operation of other service, while that of the sender-receiver is realized through multicast message delivery. No custom definition of interaction scenarios is possible in either product.

Protocol Support: Both products under evaluation provide unique services that hide the transport protocol and networking technologies and allow the inclusion of additional ones without modifications at the application level. In AUTOSAR, this is realized with a group of modules called communication services, and an Interface-Protocol Data Unit (I-PDU protocol) [40]. These concepts provide an interface to the communication network, API for network management and diagnostics, and hide protocol and network-level details from applications. Similarly, LISA has developed a proprietary module called the Media Module. This module abstracts different protocols and network types, such as Ethernet, Socket, and W-LAN, and enables uniform transmission of messages over a heterogeneous network environment.

Protocol translation: AUTOSAR and LISA provide middleware-specific, communication protocols to services, and, during message exchange, this is the only protocol services are aware of. Internally, middleware translates this, into a protocol specific to, e.g., a physical network through which the message is transported. In the case of AUTOSAR, the Communication Services pack and unpack messages to and from the I-PDU, which are then passed to network specific modules for transmission over the physical network. Likewise, LISA uses Media Module and its protocol at communication endpoints, but translates it to the network specific protocols used during transmission.

Based on analysis, extent of support the feature provides to developers is evaluated and summarized in Table V.

B. Language Entities Influenced by Variants

To understand the influence of variations in feature implementation on DSL for choreography modeling, we studied a meta-model proposed in [37]. This resulted in the identification of language entities whose implementation varies depending on the extent of the support feature provides to developers. To express variations in language entity implementation, we used language constructs such as sub-entity, attribute, and relationship multiplicity. Identified entities are as follows:

Participant: an entity that represents any logical encirclement within the system that has a degree of autonomy, and provides functionality for other Participants in the system. An example can be an accounting unit within an enterprise, a braking subsystem in the car, or a home subscriber server in telecom network. From implementation point of view, a Participant can encompass a component, collection of components, or an entire application.

To access a Participant's functionality different interfacing technologies are offered and these should be supported by middleware so that Participants can seamlessly interact. In Table II, we proposed implementation variants for a functionality access feature that can influence how a Participant, as a language entity, is implemented.

In the case of AUTOSAR, due to the use of unique and standardized interface across industry sector, Participant entity should define the attributes that are needed to describe the AUTOSAR interface only. In LISA, no structured description for accessing functionality is defined. IN this case, a Participant should include attributes that describe proprietary, LISA-specific invocation methods. In the case that a middleware product supports different interfacing technologies, a Participant entity should implement distinct sub-entity types with attributes specific to each of the supported technologies. The relationship between the Paritcipant and sub-entity should be constrained to a one-to-one relationship.

Implementation of a Participant entity is also dependent on the location transparency feature. In Table II, we proposed variations for this feature, which we see as influential for an entity implementation. Since AUTOSAR and LISA use logical names for accessing the service functionality, in both cases, a participant should provide attributes where these names will be recorded.

Role: an entity that represents the responsibility of the Participant in the scenario, and as a choreography language entity, it is a part of the participant. One Participant can have different Roles in different interaction scenarios. An example can be a Role of the organization unit that participates in choreography as "buyer" in one and "seller" in another scenario or a Role of the car engine control, which can be a "manager" in one, and a "data provider" in another scenario.

From an implementation point of view, a Role can be identified with one or more functionalities offered by Participant. Therefore, a Role must implement sub-entities for describing each of the functionalities that are included in it. Since a Participant can use different interface technologies, a set of dedicated sub-entity types should be defined, where each type would specify attributes for describing functionalities according to each of the supported technologies. In case of AUTOSAR, a Role entity should consist of functional descriptions defined according to the AUTOSAR interface. In the case of LISA, a Role should describe the functionalities based on LISA's proprietary technology for accessing applications.

TABLE V MIDDLEWARE EVALUATION RESULTS

Question	Feature	AUTOSAR	LISA
Invocation support	Functionality access	2	0
	Location transparency	1	1
	State information	1	1
Message support	Message format	2	1
	Message transformation	2	0
	Interaction scenario	1	0
Message transmission	Protocol support	2	2
	Protocol translation	2	2

Interaction: an entity that represents the exchange of information between two Roles. Exchange of information here is used to denote the ordering of one or more message exchange occurrences that together realize the Interaction. Call control application, for example, can have a Role of a service provider and must interact with the verifier, which is the Role of the subscriber information repository, to verify that a certain subscriber can use the call service. This Interaction can be realized with two message exchanges that occur in a predefined order. First, a provider sends the message with subscriber info to the verifier. Second, the verifier processes the message and sends the response back to the provider.

From an implementation point of view, an Interaction describes the order of message exchange occurrences between Roles. When implemented in language, Interaction is expressed in terms of generic (or predefined) message exchange scenarios. The idea behind this is that all message exchange scenarios conform to a single, or a combination, of generic exchange patterns. Therefore, Interaction entity implementation depends on which patterns are identified and used within an industry sector, and how they are supported by middleware product. In Table III, variants for Interaction scenarios are proposed.

AUTOSAR communication services recognize two generic scenarios, client-server and sender-receiver. Here, the Interaction entity should provide attributes for recording the two identified patterns. LISA offers no support for generic scenarios, an entity here can be implemented to allow unstructured textual description of message exchange ordering. These descriptions can be used to facilitate communication and analysis tasks.

Interaction entity implementation depends also on the implementation of message translation and protocol translation features. Participants engaged in interaction may require the translation of message content since the format in which the information is sent, is not always the format that the receiving Participant can process. An example can be a Participant that sends a SOAP message to the Participant that can receive only SIP messages. Middleware can provide features for message transformation, and implementation variants of this feature are proposed in Table III. Depending on how middleware implements the feature, Interaction will need to adopt accordingly.

AUTOSAR allows developers to define message transformations. The language entity, in this case, should include attributes for linking entities with defined transformations. LISA offers no such facilities. Including transformation-related data in an Interaction entity can only be used for documenting purposes.

Similarly to message transformation, Participants can use different communication protocols for message transmission. How middleware implements the protocol translation feature also influences implementation of the Interaction entity, and in Table IV, implementation variants are proposed.

AUTOSAR and LISA provide a feature for protocol translation, and in both cases, translation is hidden from (or transparent to) Participants that are interacting. This is accomplished by the translation feature which is a part of uniform communication service that is offered by both middleware products, and used by the Participant for communication. The Interaction entity therefore doesn’t need to include attributes for describing translations of the protocols.

In cases when middleware doesn’t support protocol translation, this task should be implemented by applications that realize the Participants. In cases when middleware implements distinct translation services for each protocol, the Interaction entity should include attributes for recording the details necessary for linking the entity with translation services.

Message Content Type: A message carries the information that is exchanged between the Roles. The format of those messages can be different, and each format specifies the types of data it can carry. Thus, the purpose of this entity is to describe those message formats.

This entity is part of the Interaction. How it is implemented in language, depends on the message formats it must be able to describe. For this reason, in Table III, we proposed implementation variants for message format support. In AUTOSAR, messages are standardized, and to define them, an entity should include only attributes relevant for the definition of AUTOSAR messages. In LISA, different message formats are supported. Still, due to the wrapping technology it uses, for entity implementation, only attributes for wrapper description should be included.

State Variable: Roles engaged in interaction can have different states based on the information that is exchanged. The value of this entity is predefined, and its purpose is to hold those values. An example of a State Variable can be “Verification State”. Based on interaction condition, a variable can hold one of two predefined values, “verification sent” or “send error”.

As a language entity, the State Variable entity is a part of the Role, and its implementation depends on state information provided by the middleware product. In Table II, we proposed implementation variants for the State Information feature. These variants express different types of state variables and influence the implementation of a language entity. Both AUTOSAR and LISA provide state information that is relevant for service- and session-level state descriptions. The language entity should, therefore, provide attributes with predefined values for capturing those items of information.

Channel Variable: Its main purpose is to store the information that is necessary for sending the message. Part of this information is, for example, the protocol that defines the rules that must be followed during message transmission. Since participants involved in interaction can use different protocols, middleware products should support them, if seamless message exchange is to be achieved.

As a language entity, the Channel Variable entity is part of an Interaction entity, and the protocol-related information that it will include depends on variations in protocol support of the middleware product. In Table IV, we proposed implementation variants that are derived based on the amount of protocol information middleware requires from

TABLE VI. EVALUATION SYNTHESIS SUMMARY

Identified Entities	Identified Features	Scales	Influence of Feature Implementation Variations on Language Entities		
			Sub-Entity	**Attributes**	**Relationship**
Participant	Functionality access	2	No influence	Describing standardized interface technology	No influence
		1	Distinct Type of Sub-Entity per supported interface technology	Distinct attribute set per Sub-Entity type for describing supported interface technology.	One Participant can have one interface technology
		0	No influence	Describing imposed interface technology	No influence
	Location transparency	2	No influence	Describing criteria for service selection	No influence
		1	No influence	Data for resolving service invocation	No influence
		0	No influence	Data for routing service request to provider	No influence
Role	Functionality access	2	Sub-Entity per functionality that is included in Role	Describing functionality according to standardized interface technology	One Role can have one or more functionalities
		1	Sub-Entity per functionality that is included in Role	Describing functionality according to Participant’s interface technology	One Role can have one or more functionalities
		0	No influence	Describing functionality according to imposed interface technology	No influence
Interaction	Interaction Scenario	2	Sub-Entity for custom interaction scenario	Description of custom interaction scenario	One Interaction can have one interaction scenario
		1	No influence	Attribute and predefined values for describing supported scenario	One Interaction can have one interaction scenario
		0	No influence	No influence	No influence
	Message transformation	2	No influence	Attributes for relating Interaction with transformations elements in middleware	No influence
		1	No influence	Attributes for relating Interaction with transformations elements in middleware	No influence
		0	No influence	No influence	No influence
	Protocol translation	2	No influence	No influence	No influence
		1	No influence	Attributes for relating Interaction with translation elements in middleware	No influence
		0	No influence	No influence	No influence
Msg. Content Type	Message format	2	No influence	Describing standardized message format	No influence
		1	Distinct Type of Sub-Entity per supported msg. format	Distinct attribute set per Sub-Entity type for describing supported msg. formats	Msg. Content Type have one msg. format
		0	No influence	Description of imposed message format	No influence
State Variable	State information	2	No influence	Attributes and predefined values on functional, session and service level	No influence
		1	No influence	Attributes and predefined values on session and service level	No influence
		0		Attributes and predefined values on service level	
Channel Variable	Protocol	2	No influence	No influence	No influence
		1	Distinct Type of Sub-Entity per supported protocol	Distinct attribute set per Sub-Entity type for describing protocol dependent communication. services	
		0	No influence	No influence	No influence

the Channel Variable. In both AUTOSAR and LISA’s case, protocol details are hidden from (or transparent to) the Channel Variable by providing uniform communication services. To transmit a message, only functionality name and message are required, while the protocol details are handled by the middleware service.

In Table VI, we summarized how variations in feature implementation influence on the implementation of the identified choreography language entities. Depending on the extent of support the middleware feature provides to developers, language entity will implement different combination of sub-entities, attributes, and relationship multiplicity.

IV. DISCUSSION

Implementation variations of identified middleware features can influence the implementation (or supplementation) of a DSL for choreography modeling. Accordingly, variations represent a valuable source of information that needs to be considered during DSL development. There are several reasons why the inclusion of this information in language can be beneficial from the software development point of view. The most important reasons are described in the following text.

Broadening the scope of DSL in the development process: Choreography DSL is an analytical tool that specifies the contractual agreement between different sub-systems. By including middleware-specific data into DSL, besides being analytical artifacts, specified models become implementation artifacts as well. A model’s role in implementation is best visible in the MDE approach, where a chain of model-to-model transformation events aims to end with generated source code. To facilitate seamless transformations, and be in compliance with middleware-induced assumptions, choreography DSL should include middleware-specific information as well.

Facilitation of communication: DSL for choreography modeling offers concepts and semantics that are needed for system analysts to agree on global service interactions. When DSL includes middleware-related information, completing models requires additional, technical-related, expertise. This way choreography modeling pulls together experts from different development areas who are cooperating on the same model and communicating using concepts and semantics that are imposed by the DSL.

Easier introduction of new developers: To develop applications on top of middleware, developers must learn and follow middleware-induced assumptions. This represents a cognitive burden for new developers, and makes modeling error-prone. When middleware concepts, rules, and constraints are built in DSL, following them comes naturally since the language itself guides the work with concepts and prevents the developer from breaking the middleware imposed rules and constraints.

A. Validity threats

As an approach, the FA-Screening mode has medium costs in time and resources, but carries a high risk for the confidence in findings. This is understandable, since the entire evaluation represents the subjective stance of a single evaluator, based only on public documentation analysis. To decrease the risk, several measures were applied during the research design. The first measure is related to the number of evaluators. Instead of one, our analysis procedure included five evaluators. Joint work ensured that the findings are based, not only on a single person's stance, but encompass the opinions of five persons with different backgrounds and expertise. The second measure is related to sources of data. Instead of consulting only publically available documents such as standards or vendor material, in *Step 3* we used company-specific material and an industry expert's knowledge.

The authors of this article believe that applying these measures during study design increased the study objectivity, and decreased the confidence risk related to findings. Additionally, researchers worked under NDAs to assure the confidentiality of company documentation, and the industry expert was familiar with the issues being researched and the way company-specific data will be treated. According to Miles and Huberman [41], described measures and practices should reduce the validity threats.

Additional drawback is that, during the score analysis step, we used the meta-model that assumes the usage of Web-Services. Web Services are only one of several component technologies that can be used for telecom and automotive systems development. Still, the model leaves enough space for customization, and therefore we found it to be generic enough for discussing choreographies in the context of other component technologies as well.

V. Conclusion and future work

The application of SOA in an embedded systems domain appears to continue to grow, and with it the need to model service interaction aspects is increasing. Using DSLs for modeling different system aspects has proven to be a good practice, and, with the growing adoption of MDE, their significance in development process will continue to grow. The research presented in this article supports this trend by focusing on the relations between the development of a DSL for choreography modeling and the underlying middleware.

Our findings suggest that the implementations of identified choreography language entities can differ depending on how middleware features are implemented. In Table VI, we describe an explanation of how this can be done. The same table can be used to answer the research question stated at the beginning. In short, based on feature implementation variations, identified language entities, which are Participant, Role, Interaction, Message Content Type, State and Channel Variable, will be implemented using different combinations of language constructs such as sub-entities, attributes, relationships, and value constraints. Concrete instances of sub-entities, attributes, and values are specific to industry sector, underlying middleware, and feature implementation technology and therefore not discussed in this article.

Future Work: The derived list of middleware features is certainly not complete. Additional features that are relevant for choreography modeling can be proposed, for example the feature for security issues. Furthermore, middleware analysis is not sufficient for DSL specification. Other problems and solution space artifacts should be analyzed to provide the needed expressiveness of the DSL. Lastly, a case study to collect broader opinions and suggestions from industry experts regarding Choreography DSL should be conducted. Future work will, therefore, continue in the above mentioned directions.

Acknowledgement

This research has been supported by ITEA2 and TEKES. The authors are grateful to AMALTHEA partners for their cooperation and to Sanja Aaramaa, Markus Kelanti, and Jarkko Hyysalo for their comments and suggestions.

References

[1] D. Krafzig, K. Banke, and D. Slama, Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall Profesionall, 2005.

[2] A. Scholz, et al. "∈ SOA-Service Oriented Architectures adapted for embedded networks," 7th IEEE International Conference on Industrial Informatics, IEEE, June 2009, pp. 599-605.

[3] L. Bocchi , J. L. Fiadeiro, and A. Lopes, "Service-oriented modelling of automotive systems," 32nd Annual IEEE International Computer Software and Applications Conference, IEEE, July 2008, pp. 1059-1064.

[4] T. Blum, N. Dutkowski, and S. Magedanz, "Evolution of SOA concepts in telecommunications," Computer, vol. 40, no. 11, Nov. 2007, pp. 46-50.

[5] R. Dijkman and M. Dumas, "Service-oriented design: A multi-viewpoint approach," International journal of cooperative information systems, vol. 13, no. 4, Dec. 2004,

pp. 337-368.

[6] C. Peltz, "Web services orchestration and choreography," Computer, vol. 36, no. 10, Oct. 2003, pp. 46-52.

[7] M. Fowler, Domain-specific languages. Addison-Wesley Professional, 2010.

[8] F. Leymann, "Web Services Flow Language (wsfl 1.0)," IBM Software Group, May 2001, [Online]. Available: http://cin.ufpe.br/~redis/intranet/bibliography/standards/leymann-wsfl01.pdf, [Retrieved: Jun, 2013]

[9] WfMC, "Process Definition Interface - XML Process Definition Language," Ver. 2.2, WfMC Standard, Doc. Number WfMC-TC-1025, Aug. 2012.

[10] OASIS, "Web Services Business Process Execution Language Ver. 2.0," OASIS Specification Draft, Aug. 2006

[11] OASIS "ebXML Business Process Specification Schema Technical Specification v2.0.4," OASIS Standard, Dec. 2006

[12] OMG, "Business Process Model and Notation", Ver. 2, Object Management Group specification, Jan. 2011.

[13] J. Zaha, A. Barros, M. Dumas, and A. T. Hofstede, "Let's dance: A language for service behavior modeling," On the Move to Meaningful Internet Systems: CoopIS, DOA, GADA, and ODBASESE, Springer, Nov. 2006, pp. 145-162.

[14] W3C "Web services choreography description language Ver. 1.0," W3C candidate recommendation, Nov. 2005.

[15] G. Decker, O. Kopp, F. Leymann, and M. Weske, "BPEL4Chor: Extending BPEL for Modeling Choreographies," IEEE International Conference on Web Service, IEEE, July 2007, pp. 296-303 .

[16] A. Barker, C.D. Walton, and D. Robertson, "Choreographing web services," IEEE Transactions on Services Computing, vol. 2, no. 2, June 2009, pp. 152-166.

[17] G. Bond, E. Cheung, and I. Fikouras, "Unified telecom and web services composition: problem definition and future directions," Proceedings of the 3rd International Conference on Principles, Systems and Applications of IP Telecommunications, ACM, July 2009, pp. 1-12.

[18] A. Iwai, N. Oohashi, and S. Kelly, "Experiences with automotive service modeling," Proceedings of the 10th Workshop on Domain-Specific Modeling, ACM, Oct. 2010, pp. 1-6.

[19] L. Lin and P. Lin, "Orchestration in Web Services and real-time communications," IEEE Communications Magazine, vol. 45, no. 7, July 2007, pp. 44-50.

[20] W3C, "Voice Browser Call Control: CCXML Version 1.0," W3C recommendation, July 2011.

[21] W3C, "State Chart XML (SCXML): State machine notation for control abstraction," W3C working draft, Aug. 2013.

[22] K. Vandikas and J. Niemoeller, "SCALE - A language for dynamic composition of heterogeneous services," Ericsson AB, Nov. 2010, [Online]. Available: http://www1.ericsson.com/res/thecompany/docs/journal_conference_papers/service_layer/101215_scale.pdf, [Retrieved: Jun, 2013]

[23] A. J. Paulo, A. Baravaglio, M. Belaunde, P. Falcarin, and E. Kovacs, "Service Creation in the SPICE Service Platform," Proceedings of the 17th Wireless World Research Forum Meeting, Heidelberg: Wireless World Research Forum, Nov. 2006, pp 1-7.

[24] J. Fiadeiro , A. Lopes, and L. Bocchi, "A formal approach to service component architecture," Web Services and Formal Methods, Springer, Sept. 2006, pp. 193-213.

[25] J. Fiadeiro, A.Lopes, L.Bocchi, and J.Abreu, "The Sensoria Reference Modelling Language," Rigorous Software Engineering for Service-Oriented Systems, Springer, 2011, pp. 61-114.

[26] B. Tsai, W.T. Huang, Q. Chen, Y. Paul, and R. A. Xiao, "SOA collaboration modeling, analysis, and simulation in PSML-C," IEEE International Conference on e-Business Engineering, IEEE, Oct. 2006, pp. 639-646.

[27] The official website of AMALTHEA project, [Online]. Avaliable: itea2.org/project/index/view/?project=10015, [Retrieved: Jun, 2013].

[28] M. Mernik, J. Heering, and A.M. Sloane," When and How to Develop Domain-Specific Languages," ACM Computing Surveys, vol. 37, Dec. 2005, pp. 316-344.

[29] S. Vinoski, "An overview of middleware,"Reliable Software Technologies-Ada-Europe, Springer, June 2004, pp. 35-51.

[30] D. C. Schmidt, "Guest Editor's Introduction: Model-driven engineering," Computer, vol. 39, no. 2, Feb.2006, pp. 25-31.

[31] E. Di Nitto and D. Rosenblum, "Exploiting ADLs to specify architectural styles induced by middleware infrastructures," Proceedings of the 21st International Conference on Software engineering, ACM, May 1999, pp. 13-22.

[32] B. Kitchenham, S. Linkman, and D. Law, "DESMET: a methodology for evaluating software engineering methods and tools," Computing & Control Engineering Journal, vol. 8, no. 3, June 1997, pp. 120 - 126.

[33] V. R. Basili, G. Caldiera, and H.D. Rombach, "The Goal/Question/Metric approach," Encyclopedia of software engineering, John Wiley & Sons, Inc, 1994, pp. 528-532.

[34] L. Aversano, G. Canfora, A. De Lucia, and G. Pierpaolo, "Business process reengineering and workflow automation: a technology transfer experience," Journal of Systems and Software, vol. 63, no. 1, July 2002, pp. 29-44.

[35] The official website of AUTOSAR, [Online]. Available: http://www.autosar.org/, [Retrieved: March, 2013].

[36] P. Berander and P. Jonsson, "A goal question metric based approach for efficient measurement framework definition," Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, ACM, Sept. 2006, pp. 316-325.

[37] W3C, "WS choreography model overview," W3C working draft, Mar ch 2004.

[38] AUTOSAR GbR, "Software Component Template," Ver. 4.2.0, R4.0 Rev 3, AUTOSAR Standard, 2011.

[39] AUTOSAR GbR, "Specification of RTE," Ver. 3.2.0, R4.0 Rev 3, AUTOSAR Standard, 2011.

[40] AUTOSAR GbR, "Specification of Communication," Ver. 2.0.1, AUTOSAR Specification, 2007.

[41] M.B. Miles and M.A. Huberman, Qualitative data analysis: a sourcebook of new methods. Sage, 1984.