A simple classification of discrete system interactions and some consequences for the solution of the interoperability puzzle

Johannes Reich [*] **Tizian Schröder** [**]

[*] *SAP SE, 69190 Walldorf, Germany (e-mail: johannes.reich@sap.com)*
[**] *Otto von Guerike Universität, 39106 Magdeburg, Germany (e-mail: tizian.schroeder@ovgu.de)*

Abstract: In this article, we introduce a classification of system interactions to guide the discourse on their interfaces and interoperability. It is based on a simple, but nevertheless complete classification of system interactions with respect to information transport and processing. Information transport can only be uni- or bidirectional and information processing is subclassified along the binary dimensions of state, determinism and synchronicity.
For interactions with bidirectional information flow we are able to define a criterion for a layered structure of systems: we name a bidirectional interaction ”horizontal” if all interacting systems behave the same with respect to state, determinism and synchronicity and we name it ”vertical” — providing a semantic direction — if there is a behavioral asymmetry between the interacting systems with respect to these properties.
It is shown that horizontal interactions are essentially stateful, asynchronous and nondeterministic and are described by protocols. Vertical interactions are essentially top-down-usage, described by object models or operations, and bottom-up-observation, described by anonymous events.
The interaction classification thereby helps to better understand the significant relationships that are created between interacting discrete systems by their interactions and guides us on how to talk about discrete systems, their interfaces and interoperability.
To show its conceptual power, we apply the interaction classification to assess several other architectural models, communication technologies and so called software design or architectural styles like SOA and REST.

Keywords: horizontal interaction, vertical interaction, interface, protocol, software layering

1. INTRODUCTION

Civil, mechanical, electrical, or software engineering - they all deal with the design of systems. The term ”system engineering” was coined in the Bell Laboratories in the 1940s (e.g. Buede (2011)). Especially with the advent of cyber physical systems also the software engineering discipline articulates the importance of a unifying view (e.g. Sifakis (2011)).

The base for such a unifying view is the common notion of a system based on its functionality, that is its transformational behavior — a notion that is tightly related to the concepts of computability and determinism of computer science.

Although systems can be described as isolated entities, separating the world into itself and the rest, their main purpose is to interact. Hence, the notion of an interface as the representation of a system with respect to its interactions with other systems comes to the fore.

As our conception of a given system is only well defined as we know its transformational behavior, the same is true for its interfaces. Knowing the interfaces of some systems should make the question, whether these systems can interoperate, decidable. To ”interoperate” means that the information processing in all involved systems is accomplished in a way that, by interaction, a specified purpose is fullfilled (IEC, 2001ff).

So far, so good — despite these considerations, the enormous growth of the number of interacting electronic devices in the last two decades, the internet, had been mainly due to semantically [1] agnostic information transport protocols like HTTP, FTP, SMTP, etc., leaving the essential problems of interoperability within the sphere of the human mind.

In the meantime, however, technical information-processing systems are increasingly being integrated into electronic interactions on a content-level with a certain degree of autonomy. Automation in particular, as the technology domain that deals with the construction of processes or procedures that are performed with minimal human assistance, has a great interest in clear and systematic concepts of semantic interoperability.

But on the contrary, a plethora of different technologies has evolved, each with the promise to solve some aspects of the interoperability puzzle. Especially for engineers not from computer science this makes it quite difficult to create interoperable components with the required minimal effort. The different underlying models, in conjunction with different terminolo-

[1] We use the term ”semantic” as a synonym for ”with respect to processing” to emphasize the distinction between information transport and information processing. In this sense, we say that information gets transported and the meaning of the information, its significance, is attributed by proccessing within interactions. We speak more concrete of an ”interaction semantics”.

gies, or — even worse — with the same terminology, hinder mutual understanding and drive the costs for successful device integration. Roughly speaking, there is no mutual agreed language to talk about interoperability, perhaps best exemplified by the research gap we are addressing in this paper that despite its widespread use of layer models in software engineering, a clear ordering criterion for software components seems to be lacking.

The contribution of this paper is a simple classification of discrete system interactions which can be viewed as an extension of the OSI-model (ITU-T, 1994) based on sound semantic principles. It implies a clear ordering criterion for software components — in fact, there are two tightly related ones. And it makes it possible even for a non-software engineer to avoid some fundamental mistakes in interoperability design. In fact, we view it as a contribution to a reference model of interaction semantics in the sense of a "conceptual framework for understanding significant relationships among the entities of networking systems, based on a small number of unifying concepts" (MacKenzie et al., 2006).

Based on a proposal of one the authors, it has already been adopted by the German VDI/VDE Fachausschuss GMA 7.20 for their VDI-guideline 2193 (VDI, 2019) and the German BITKOM AK Interoperabilität for their recent whitepaper to provide guidance for the German industry on semantic interoperability (Bitkom, 2020). One of its main conclusions that horizontally interacting systems do so in a nondeterministic, stateful, and asynchonous way strongly influenced the initiation of the project "Verwaltungsschale vernetzt" (Diedrich, 2019).

The structure of the article is as follows. In the second section we lay out the formal definitions of the basic notions which we need to build our reference model. In the third section we introduce a simple classification of system interactions based on information flow and processing. In the forth section we use our reference model to discuss other state of the art reference models, so called architectural styles and interaction supporting technologies. And in the last section, we summarize a couple of direct consequences of our reference model.

2. THE MODEL OF SYSTEM INTERACTION

In this section we provide formal definitions for the basic notions that our interaction classification is based upon. Due to the scope of this article, we will use them only to make the meaning of our classifcation more explicit and not to derive more intricate relationships between them (For more details, see e.g. Reich (2016/2020)).

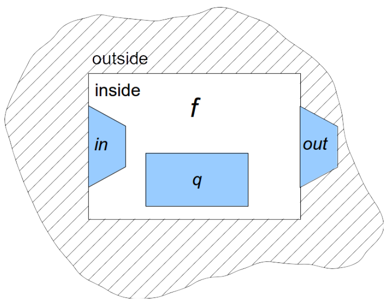

Figure 1. A symbolic representation of a system with its three signals $(in, out, q) : T \to Q \times I^\epsilon \times O^\epsilon$ and its system function f, separating the inside from the outside.

There seems to be a consensus (e.g. IEC (2001ff)) that a system separates an inside from the rest of the world, the environment (see Fig. 1). We define what we call a discrete multi-input system to consist of a set of discrete time values T together with a time function $succ$ which always provides the next point in time; three signals, an input signal in, an output signal out and an internal signal q, each mapping time onto the respective alphabet I, O, and Q; and a system function f mapping input and internal signal values at time t onto output and internal signal values at time $t' = succ(t)$. The system is called a multi-input system because it could well be that the input characters are vectors, where some components for some characters remain empty. With the convention that ϵ is the empty character and for any alphabet A we write $A^\epsilon = A \cup \{\epsilon\}$, we therefore have the following formal definition:

Definition 2.1: A discrete system $\mathcal{S}$ is defined as $\mathcal{S} = (T, succ, Q, I, O, q,\ in, out, f)$. Q, I and O are alphabets, whereas only Q has to be non-empty. The signals $(q, in, out) : T \to Q \times I^\epsilon \times O^\epsilon$ form a discrete system for the time step $(t, t' = succ(t))$ if the partial function $f : Q \times I^\epsilon \to Q \times O^\epsilon$ with $f = (f^{int}, f^{ext})$ maps as following

$$\begin{pmatrix} q(t') \\ out(t') \end{pmatrix} = \begin{pmatrix} f^{int}(q(t), i(t)) \\ f^{ext}(q(t), i(t)) \end{pmatrix}.$$

If the input signal contains two or more components, we also call it a "multi-input system" (MIS).

We can describe the behavior of a discrete system with an I/O-transition system. Thereby we get rid of the explicit time dependency, exploiting its stepwise character. Additionally we are able to look at the behavior of a discrete system in the sense of a projection, which becomes essential for protocols.

Definition 2.2: An I/O-transition system $\mathcal{A}$ is defined as $\mathcal{A} = (Q, I, O, q_0, \Delta)_{\mathcal{A}}$. $Q_{\mathcal{A}}$, $I_{\mathcal{A}}$ and $O_{\mathcal{A}}$ are alphabets, whereas only Q has to be non-empty. $q_0 \in Q$ is the initial value and $\Delta_{\mathcal{A}} \subseteq I^\epsilon \times O^\epsilon \times Q \times Q$ is the transition relation.

Such an $\mathcal{A}$ describes the behavior of a discrete system $\mathcal{S}$ in the sense of a projection, if $Q_{\mathcal{S}} \subseteq Q_{\mathcal{A}}$, $I_{\mathcal{S}} \subseteq I_{\mathcal{A}}$, $O_{\mathcal{S}} \subseteq O_{\mathcal{A}}$, $q(0) = q_0$ and a projection function [2] $\pi = (\pi_Q, \pi_I, \pi_O) : Q_{\mathcal{S}} \times I^\epsilon_{\mathcal{S}} \times O^\epsilon_{\mathcal{S}} \to Q_{\mathcal{A}} \times I^\epsilon_{\mathcal{A}} \times O^\epsilon_{\mathcal{A}}$ exists and $\Delta_{\mathcal{A}}$ is the smallest possible set, such that for all times $(t, t' = succ(t)) \in T_{\mathcal{S}} \times T_{\mathcal{S}}$, and for all possible signal runs it holds $(\pi_Q(q(t)), \pi_Q(q(t')), \pi_I(in(t)), \pi_O(out(t'))) \in \Delta_{\mathcal{A}}$.

We can say that our model of information processing treats information as characters of an alphabet and presupposes a processing context in the form of an I/O-transition relation.

As is illustrated in Fig. 2, interaction between discrete systems means that they share a common signal: The output signal of a "sender" system is identical with the input signal of a "receiver" system. Thereby the output value of a transition of a "sender" system serves as the input value of a transition of a "receiver" system. In other words, interaction means that information is transported and the execution schema of the product automaton has to be modified such that a 'transported' character has to be processed next.

The interaction mechanism is thereby formally based on identically named input and output characters of otherwise anonymous transitions [3] .

[2] A projection function π fulfils the property $\pi = \pi \circ \pi$.

[3] There are many interaction models based on transition systems with named transitions, where the coupling of different systems is achieved by identically

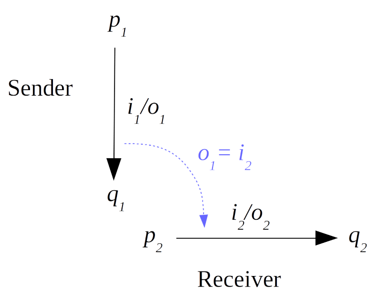

Figure 2. Interaction between two systems where the output character of a sender system is used as an input character of the receiving system.

For our classification of system interaction in the next section, we define the following behaviors for sender and receiver systems.

- A sending system behaves synchronously if the completion of the receiver's transition is a necessary and sufficient requirement for the sending system's next transition. Otherwise it behaves asynchronously.
- A receiving system behaves deterministically if its transition relation represents a transition function. Otherwise it behaves nondeterministically
- A receiving system behaves statefully if its set of internal state values Q has more than one element. Otherwise it behaves stateless.

Please note that in our definition, synchronicity is a property of the sending system. That is, whether a sender waits or not (behaves synchronously or asynchronously) is not detectable by the receiver, processing any received character.

Whether an interaction is deterministic or nondeterministic influences the kind of goal of the interaction. In the first case, the goal of the interaction is a superordinated function, leading to system composition (Reich, 2016/2020). In the second case, without creation of a superordinated system function, the need to distinguish desirable from non-desirable behavior necessitates an additional acceptance component, extending the I/O-transition systems to become I/O-automata or transducers, leading to protocols.

3. CLASSIFICATION OF SYSTEM INTERACTION

The essential idea (Reich, 2015) is that system interactions can be classified along the two relevant aspects of interoperability: information transport and information processing. Information transport can be either unidirectional or bidirectional. Information processing is further subclassified along the three subdimensions state, determinism and synchronicity (see section 2).

We have chosen these three subdimensions of information processing because of their direct influence on the form of the appropriate interface. For example: there are no return parameters in the asynchronous case. In the deterministic case, it is possible to represent the intended input-output-relation of a system by an operation, mapping state values onto state values. With statefulness and determinism the desired functionality can be described object oriented and last but not least, we describe stateful, nondeterministic and asynchronous interactions as protocols.

This classification is complete in the sense that every interaction, that can be described with the formalism of section 2, can be classified accordingly.

3.1 Interaction with unidirectional information flow

As backward communication is irrelevant for unidirectional interfaces, we can disregard any synchronicity. The two most important classes are:

Deterministic: We name a sequence of systems a "pipe", where an overall computational function is computed in a number of successive steps on a "data flow", where the input of each pipe component is the output of the predecessor component (except for the first one). Thereby pipes provide the means for sequential and parallel system coupling (Reich, 2016/2020). To be complete, a pipe mechanism must be able to fork and join pipes.

Nondeterministic: We name an interaction between a sender system and a receiver system an "observation" if the sender system makes no assumptions on the determinism and statefulness of the receiver system.

3.2 Interaction with bidirectional information flow

As it is bidirectional, the flow of information as such does not determine any direction of the interaction relation any longer, but the direction is determined by the way, the information is processed in all interacting systems - and therefore is a semantic property. We distinguish two main bidirectional interaction classes:

Horizontal interaction: All interacting components behave the same with respect to the three semantic sub-dimensions, i.e. there is a behavioral symmetry. Only the combination stateful, nondeterministic and asynchronous behavior makes sense. Mutual determinism results in a deadlock where each system waits for some input. Nondeterminism and statelessness implies randomness. And mutual synchronous behavior makes only sense in the calculation of recursive functionality.

Horizontal interactions are described by bi- or multilateral interfaces of protocol roles. This multilaterality manifests itself by the fact that the knowledge of all roles of a protocol is necessary to guarantee important properties of this form of interaction.

Vertical interaction: the interacting components behave differently with respect to the three semantic subdimension. The resulting asymmetric setting creates a semantic direction of the interaction. The "lower" component behaves deterministically and can therefore be described by a function call (with exceptions). We say that it does not make any assumptions with respect to the behavior of the "upper" component and can therefore provide information upwards only by an event-mechanism that is similar to the observation in the case of unidirectional information flow. Actually, within such an interface, only the lower component is described with its functionality and its events — why we call these interfaces "unilateral".

(or complementarily) named transitions, e.g. Hoare (1985/2004); Milner et al. (1992).

3.3 Components and their hierarchies

Components: An important concept that is touched by our classification is that of components. Components are supposed to be building blocks which easily fit together (e.g. Heineman and Councill (2001); Szyperski (2002)).

Our classification suggests to use both, the characteristics of a system's I/O relation as well as its composition behavior, to define the component concept. Two systems might comprise the same I/O-relation but may differ in their composition behavior like operations versus pipes. Both are deterministic (according to our definitions), but operations provide their output back to the component where they received their input from, while pipe components provide their output to the ”next” component of the pipe.

Bidirectional interactions creating complex recursive functionality do not follow any simple composition scheme and should be avoided on the level of components. Thereby, from an software engineering point of view, components also mark a systematic border of design complexity, where any functionality that is created by general recursion moves into the component's innards.

Layers: Our rather simple classification allows the definition of a layering in a component based system and thereby relates interoperability to these different layers. Components that interact vertically belong to different layers, components that interact horizontally belong to the same layer. Observed components belong to lower layers and all components of a pipe belong to the same layer.

Please note, that one has to be precise to what kind of hierarchical relation one refers. To illustrate the problem, we provide a simple example. Fig 3 shows a simple system composition where three systems $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$ compose to a supersystem $\mathcal{S}$ with system function $f_{\mathcal{S}}(x) = 2x+5$. System $\mathcal{S}_2$ contributes its system function $f_{\mathcal{S}_2}(x) = 2x$, $\mathcal{S}_3$ contributes $f_{\mathcal{S}_3}(x) = x+5$, and System $\mathcal{S}_1$ is a multi-input system which mostly coordinates system $\mathcal{S}_2$ and $\mathcal{S}_3$ in a non-trivial recursive manner. As we can easily see, there is no interaction between the subsystems and their supersystem, but instead, the supersystem is created by the deterministic interactions of the subsystems.

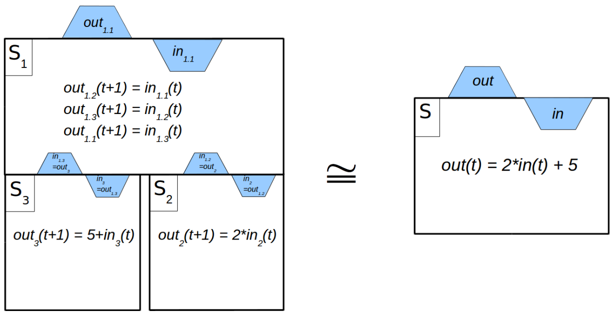

Figure 3. Three systems $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$ compose to a supersystem $\mathcal{S}$ with the overall (super)system function $f_{\mathcal{S}}(x) = 2x + 5$.

According to our classification, system $\mathcal{S}_1$ interacts hierarchically with both $\mathcal{S}_2$ and $\mathcal{S}_3$ as it determines their transitions by its output and not vice versa. Thereby it is justified to say that because of their interaction, system $\mathcal{S}_1$ belongs to a higher layer as both, systems $\mathcal{S}_2$ and $\mathcal{S}_3$. The supersystem $\mathcal{S}$ is not mentioned at all in this description. This is shown in the left side of Fig. 4.

However, we could also define a hierarchical relation in the sense of ”is-part-of” where both subsystems, $\mathcal{S}_2$ and $\mathcal{S}_3$, are part of the supersystem $\mathcal{S}$, which is shown in the right side of Fig. 4. Now, system $\mathcal{S}_1$ is not mentioned any longer.

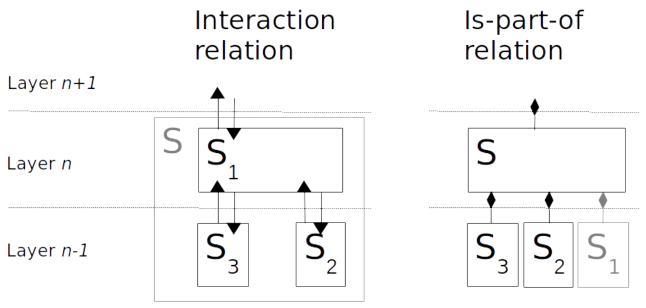

Figure 4. Due to their interactions, the systems of Fig. 3 can be ordered in two different ways. On the left side, they are ordered according to their interaction relation. The arrows represent the information flow. The supersystem $\mathcal{S}$ is only sketched in light gray to show that it does actually span several layers in this hierarchy. Or they can be ordered according to their ”is-part-of”-relation, shown on the right side. Now it is the supersystem $\mathcal{S}$ that is super ordinated. The relation is represented by a filled diamond and a solid line. There is no information flow between the layers.

This latter hierarchy is used in imperative programs and the object oriented world with their method-construct. A method represents a function which — if not elementary — depends on other methods. Thus, with methods we do not follow an interaction-oriented interface concept, but a structur-oriented interface concept, supporting the ”is-part-of” relation.

The formal test for the claim, that a component can be put into a certain layer is to provide a unilateral interface with generic events and operations and to assure that the component itself uses only operations and reacts to generic events of components of lower layers.

With the ”is-part-of”-relation, the higher the layer, the more abstract — or the less technical — is the level of information processing.

Remote operations Remote operations take advantage of the fact that a ”remote” operation can be partitioned into a sequence of concatenated operations of serialization, deserialization, transport and local processing.

In the case of a remotely used object, the communication components of both sides become a ”communciation layer” that can indeed be hidden behind the remote operation's signature — adding, however, remote exceptions. Thus, despite the similarity between a local and a remote object usage, the imponderables of information transport usually introduces non-determinism. Due to their (much) higher unreliablity, remote operations should not be used to change remote state.

Protocols A protocol (see e.g. Holzmann (1991) for an overview) is best understood as a signaling game in the sense of P. Grice (Grice, 1989) where all participants are only described as a ”role” in the sense of a projection, giving each other mutual hints and rely on the fact that their sent information will be appropriately processed. In our model, each participating system can be described in its participating role by an automaton according to definition 2.2. A protocol has to be ”wellformed” in the sense that for all sent characters there has to be a receiving transition, ”complete” such that there are no other inputs, ”interuptable” in the sense that it does not have infinite chains of interactions, and ”consistent” such that it should be possible to meet the acceptance condition from each reachable state.

As P. Grice pointed out we can distinguish between an assumed receiver semantics of the sender and the actual semantics of the receiver. As both relate to the processing of the receiver, they can easily be compared.

We think that extending the interaction description of protocols by ”decisions” as an additional internal input alphabet to fill in the non-determinism leads to the game-notion (Reich, 2009).

4. STATE OF THE ART AND APPLICATION OF THE REFERENCE MODEL

In this section we use our reference model to discuss other state of the art reference models, so called architectural styles and interaction supporting technologies.

4.1 The Open Systems Interconnection (OSI) basic reference model

The Open Systems Interconnection (OSI) basic reference model (ITU-T, 1994) was very influential, as it established the notion of a layered software architecture.

However, the OSI model assumption, that ”OSI is concerned with the exchange of information between open systems (and not the internal functioning of each individual real open system).” turns out to be inconsistent. One cannot refrain from saying anything about the structure of information processing and at the same time making claims about its inner structure, like layering. Also, the OSI model was not sufficiently precise with respect to the kind of hierarchy. And based on these inconsistencies, the OSI-assumption that the information processing between protocol-connected components always happens on equal layers proved to be wrong with remote function calls.

Thus, our classification provides a formal justification of the intuition of the OSI-model to view software applications as being layered. Thereby, our model also explains, why the OSI-model found its way into reality only up to its 4-th layer, as the management of a ”session” state cannot be attributed to a dedicated layer in the general case. Only in the case of vertical interaction, the interaction related state can be encapsulated into a state of an intermediate ”session layer”. In the case of horizontal interaction, the interaction related state genuinely belongs to the components of the same semantic layer that mutually interact.

4.2 Examples of inadequate usage of the layer concept in models of interoperability

We give three state of the art examples where the lack of a consistent ordering criterion leads to an inconsistent layer concept. As a result categories are mixed and these models provide at most a more intuitive level of understanding, considerably limiting their applicability for the engineer.

One example is the ”Level of Conceptual Interoperability Model (LCIM)“ (Tolk et al., 2006), which consists of the 7 alleged layers: no [interoperability], technical, syntactic, semantic [defined not in our sense], pragmatic, dynamic, and conceptual interoperability. Obviously, it is not interaction which accounts for this hierarchy, but what else? Even for the technical information transport e.g. by the internet protocol, a certain structure (=syntax) of the transported information is necessary. It is unclear how to separate semantic from pragmatic aspects. For example, how can the meaning of a bank tranfer be described without refering to some action a bank is supposed to take? Etc.

Another example is the architecture axis of the three-dimensional Reference Architecture Industry 4.0 (RAMI4.0) (DIN, 2016). It consists of the 6 alledged layers: asset, integration, communication, information, functional and business. Again it is not interaction that accounts for this hierarchy. And no other criterion is given.

A third example is the IIC Connectivity Framework (Industrial Internet Consortium). It builds upon parts of the LCIM. The transport layer is supposed to achieve technical interoperability and a framework layer is supposed to achieve syntactical interoperability in the sense of providing all means to exchange structured data. However, determining the type of the exchanged data also determines whether it is to be understood as an operation, as a business document or a generic event — requiring an agreement on the interaction class beforehand. As a result, the authors neglect to discuss how to support the important class of horizontal interactions.

In effect, most of these so called ”layers” are in fact aspects, that is, something that can be described from a certain point of view in the sense of a projection. Thus, we have data aspects, communication aspects, functional aspects, etc. But all these aspects overlap and do not constitute a hierarchical division of the thing of interest.

4.3 Interaction pattern based approaches

There exist quite a few approaches that present themselves as ”pattern based” and often claim to be motivated by the speech act theory (Austin, 1962). Examples are the UN/CEFACT modelling methodology (UN/CEFACT, 2003) or RosettaNet (RosettaNet, 2001) using ”transaction patterns”. A similar concept had been the ”message exchange pattern” (W3C, 2007) of the service oriented architecture (SOA).

The application of our classification already fails at the very beginning, as these approaches do not relate sufficiently to the transformational behavior — the processing capabilities — of the interacting systems. Without referring to the transition relation of sender and/or receiver, the semantics (=processing) of any ”interaction pattern” is illdefined.

An example for a total lack of reference to the transformational behavior was SOA. The idea of a SOA was introduced by Roy W. Schulte and Yefim V. Natis of Gartner in 1996 (Schulte and Natis, 1996) and it was endorsed by virtually all large IT companies like Microsoft, IBM, SUN, Oracle, Adobe, SAP, etc. Within the OASIS SOA reference model (MacKenzie et al., 2006), a service is defined vaguely as a ”Mechanism to enable access to one or more capabilities where the access is provided by a prescribed interface and is exercised consistent with constraints and policies as specified by the service description." At no place within SOA the service definition refers to some transformational behavior, making any distinction between deterministic vs. nondeterministic interaction impossible. But, from a syntactic point of view, SOA ties oneself down to (remote) objects with methods. Thus, well defined services specified by a WSDL-specification can only represent accessible functionality — and not multilateral interfaces of protocol participants, which is the true interface of a service as it is understood by economists. For example to get a wall painted in a newly build house, one has to invite offers, accept a bid, arrange and rearrange appointments, check the result and, in case of acceptance, pay the bill and finally keep the documents for tax relief — a relation full of state, asynchrony and nondeterminism.

4.4 Representational State Transfer (REST)

REST (Fielding, 2000) can be viewed as the attempt to transfer the principles of stateless communication together with semantic agnostics - both principles of the HTTP-protocol - onto the interactions of networking applications. Currently it is often positioned as a simpler variant of SOA.

A REST-service is supposed to adhere to the principle of addressability, that each resource has to have a unique URI, and statelessness, that each REST-message is supposed to contain all the information that is necessary for the processing which it initiates. Sometimes idem potency (e.g. Pautasso (2014)) is mentioned, that the called transport methods are supposed to have an identical effect, no matter when they are called.

These "principles" are in direct contradiction to the proposed interaction model where horizontal interactions usually are stateful and the exchanged information is usually not processed in an idempotent way. From the perspective of the proposed interaction model, REST is a methodological chimera with parts from the object as well as the protocol world. On the one side, it only specifies a letter-box mechanism, which gives some of the letter life cycle functionality to the sender. It does not require to specify the transformation behavior of a REST-service or the relation between different REST-services. But on the other side, it requires that all resources have to be published to the public, offering a lot of otherwise private information.

4.5 Evaluating interaction supporting technologies

One strength of our interaction classification is to allow a simple assessment of whether a given interaction supporting technology supports a certain interaction class. Thereby we can specify its right domain of use and point out when it fails. First, it separates these technologies into information transport versus information processing categories, depending on whether they relate to the content of the transported information or not.

Then it makes clear that content-oriented technologies supporting only remote object models like DCOM or OPC-UA are of little value when it comes to implement protocol based interactions.

It also explains, why it is inappropriate to use technologies primarily supporting unidirectional observation like publish-subscribe (e.g. MQTT) for bidirectional horizontal interactions, as is the case in many "service-bus"-models. In a unidirectional observation the sender does not make any assumptions about the processing of the receiver. Thus, the created events either have a strongly standardized format, like instance X of type Y changed its state from Z1 to Z2 - making mutual understanding impossible. Or the created events become arbitrarily broad to contain all the information that is possibly available, just in case, the (unknown) receiver might find it necessary.

We also can derive a simple necessary requirement for any technology to support horizontal interactions, namely in order to implement protocol roles most naturally, it should support state machines and documents,

5. CONCLUSION

We introduced a rather simple classification of interactions of discrete systems. It is supposed to guide the discourse on interoperability, what the real challenges are, how to achieve them, etc. within the system engineering community for example in standardization efforts.

Our model is conceptually sparse as it uses very little ad hoc assumptions. It is based on a sound system model, fully compatible with the model of information transport and processing. And it is expressive as it entails a whole series of important practical consequences:

- An interface of a system is only so well defined as it states the transformational behavior with respect to the interaction the system is involved in. Only then we can make further statements about interoperability in a semantic sense, that is, related to the respective processing of the exchanged information.
- The relation between interacting systems is determined by their interaction and cannot be arbitrarily chosen.
- It shows the importance of a precise and verifiable order criterion for a well defined layer-structure of systems.
- It is very important to distinguish between interaction scenarios with unidirectional and bidirectional information flow. Especially, an interaction with bidirectional information flow is not just the superposition of two otherwise independent single-flow scenarios. Adding an information flow "backwards" in a formely single-flow scenario changes the game of interoperability fundamentally.
- Looking only at the act of information transport, one cannot say — per definition — anything about the processing of the information, for example, whether the processing of the information is stateful, synchronous or deterministic. Thus, terms like "message based integration" may even be misleading, as they suggest that the processing context of the transported information can be ignored.
- Any attempt to propose some inner structure of information processing systems without refering to the structure of the information processing itself is inconsistent.
- Horizontal interactions are nondeterministic, asynchronous and stateful.
- Technologies providing only access to remote functionality in the sense of object models, are of little help to implement horizontal interactions.
- To easily structure applications into layers, an internal event mechanism is needed to avoid the use of operation calls to provide information for higher level processing.

Only without circular functional dependencies does an operation call formally indicate a descent into a lower software layer.

- We have to distinguish between transformational and compositional behavior, leading to the distinction between systems and components.

REFERENCES

Austin, J.L. (1962). *How to Do Things with Words.* Cambridge (Mass.).

Bitkom (2020). *Vorschlag zur systematischen Klassifikation von Interaktionen in Industrie 4.0 Systemen – Hinführung zu einem Referenzmodell für semantische Interoperabilität.* White paper.

Buede, D.M. (2011). *The engineering design of systems: models and methods*, volume 55. John Wiley & Sons.

Diedrich, C. (2019). *Verwaltungsschale vernetzt - Interoperabilität zwischen I4.0 Komponenten.* Forschungsportal Sachsen-Anhalt.

DIN (2016). SPEC 91345:2016-04 Referenzarchitekturmodell Industrie 4.0 (RAMI4.0).

Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* Ph.D. thesis, University of California, Irvine.

Grice, H.P. (1989). *Studies in the Way of Words.* Harvard University Press.

Heineman, G.T. and Councill, W.T. (eds.) (2001). *Component-based Software Engineering: Putting the Pieces Together.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Hoare, C. (1985/2004). *Communicating Sequential Processes.* Prentice Hall.

Holzmann, G.J. (1991). *Design and validation of computer protocols.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

IEC (2001ff). IEC 60050 International Electrotechnical Vocabulary.

Industrial Internet Consortium (2018). IIC: The Industrial Internet of Things, Volume G5: Connectivity Framework.

ITU-T (1994). X.200 Information Technology - Open Systems Interconnection – Basic Reference Model.

MacKenzie, C.M., Laskey, K., McCabe, F., Brown, P.F., and Metz, R. (eds.) (2006). *Reference Model for Service Oriented Architecture 1.0.* OASIS.

Milner, R., Parrow, J., and Walker, D. (1992). A calculus of mobile processes (parts I and II). *Information and Computation*, 100(1), 1–77.

Pautasso, C. (2014). RESTful web services: principles, patterns, emerging technologies. In *Web Services Foundations*, 31–51. Springer.

Reich, J. (2009). The relation between protocols and games. In S. Fischer, E. Maehle, and R. Reischuk (eds.), *Informatik 2009: Im Focus das Leben, Beiträge der 39. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 28.9.-2.10.2009, Lübeck, Proceedings*, volume 154 of *LNI*, 3453–3464. GI.

Reich, J. (2015). Eine semantische Klassifikation von Systeminteraktionen. In D. Cunningham, P. Hofstedt, K. Meer, and I. Schmitt (eds.), *INFORMATIK 2015*, Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn, 1545–1559.

Reich, J. (2016/2020). Composition, cooperation, and coordination of computational systems. *CoRR*, abs/1602.07065.

RosettaNet (2001). *Implementation Framework: Core Specification, V02.00.00.*

Schulte, R.W. and Natis, Y.V. (1996). *"Service-Oriented" Architectures, Part 1 and 2.* SSA Research Notes SPA-401-068, -069, Gartner Group.

Sifakis, J. (2011). A vision for computer science - the system perspective. *Central European Journal of Computer Science*, 1(1), 1008–116.

Szyperski, C. (2002). *Component Software, Beyond Object-Oriented Programming.* Addison-Wesley, 1 edition.

Tolk, A., Turnitsa, C.D., Diallo, S.Y., and Winters, L.S. (2006). Composable M&S web services for net-centric applications. *The Journal of Defense Modeling and Simulation*, 3(1), 27–44.

UN/CEFACT (2003). *Modelling Methodology (UMM) User Guide, V20030922.*

VDI (2019). VDI/VDE 2193-1 — Language for I4.0 components - Structure of messages.

W3C (2007). Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language.