# A Choreography-Driven Approach to APIs: The OpenDXL Case Study

Leonardo Frittelli, Facundo Maldonado, Hernán Melgratti, Emilio Tuosto

# A Choreography-Driven Approach to APIs: the OpenDXL Case Study*

Leonardo Frittelli[1], Facundo Maldonado[1], Hernán Melgratti[2], and Emilio Tuosto[3]
{leonardo_frittelli,Facundo_Maldonado}@mcafee.com,
hmelgra@dc.uba.ar, emilio.tuosto@gssi.it

[1] McAfee Cordoba, Argentina
[2] ICC - Universidad de Buenos Aires - Conicet, Argentina
[3] Gran Sasso Science Institute, Italy & University of Leicester, UK

**Abstract.** We propose a model-driven approach based on formal data-driven choreographies to model message-passing applications. We apply our approach to the *threat intelligence exchange* (TIE) services provided by McAfee through the OpenDXL industrial platform. We advocate a chain of model transformations that (i) devises a visual presentation of communication protocols, (ii) formalises a global specification from the visual presentation that captures the data flow among services, (iii) enables the automatic derivation of specifications for the single components, and (iv) enables the analysis of software implementations.

## 1 Introduction

We propose a methodology for the modelling and analysis of (part of) OpenDXL, a distributed platform that embraces the principles of the *API-economy* [18,10]. In this context applications are services built by composing APIs and made available through the publication of their own APIs. In fact, the APIs of OpenDXL are paramount for enabling the openness of the platform, its growth in terms of services (currently the platform offers hundreds of different services), and its trustworthiness. The overall goal of OpenDXL is to provide a shared platform for the distributed coordination of security-related operations. A key aspect of the platform is to foster public APIs available to stakeholders for the provision or consumption of cyber-security services.

A well-known issue in API-based development is that APIs interoperability heavily depends on the (quality of) documentation: "An API is useless unless you document it" [30]. Proper documentation of APIs is still a problem. The current practice is to provide informal or semi-formal documentation that makes

* Research partly supported by the EU H2020 RISE programme under the Marie Skodowska-Curie grant agreement No 778233, by UBACyT projects 20020170100544BA and 20020170100086BA, PIP project 11220130100148CO, and by MIUR project PRIN 2017FTXR7S *IT MATTERS* (Methods and Tools for Trustworthy Smart Systems)

it difficult to validate software obtained by API composition, to establish their properties, and to maintain and evolve applications [2]. The OpenDXL platform is no exception. The APIs of the platform is mostly described in plain English.

We advocate a more systematic approach that, turning informal documentation of APIs in precise models, enables the application of formal methods to develop and analysis services. We focus on *threat intelligence exchange* (TIE) [24], one of the OpenDXL APIs for the coordination of activities such as assessment of security-related digital documents or reaction to indicators flagging suspicious behaviour or data. The API of TIE is part of OpenDXL and it has been designed to enable the coordination of distributed security-related activities. More precisely, TIE APIs support the management of crucial cyber-security information about assets (digital or not) of medium-size to big organisations.

Components for TIE developed by third-party stakeholders sometimes exhibit unexpected behaviour due to the ambiguity of the documentation of communication protocols. In fact, TIE relies on an event-notification communication infrastructure to cope with the high number of components and the volume of the communication. This asynchronous communication mechanism requires the realisation of a specific communication protocol (an application-level protocol) for the various components of the architecture to properly coordinate with each other. To address these issues, we propose a more rigorous approach to the development and documentation of the APIs. We adopt a recent behavioural type system [5] to give a precise model of some TIE services. Besides the resolution of ambiguities in the API documentation, our model enables some static and run-time verification of TIE services. We will discuss how these models could be used to check that the communication pattern of components is the expected one. Also, we will show how our behavioural types can be used to automatically verify logs of executions that may flag occurrences of unexpected behaviour.

*Summary of the contributions* Our overall contribution is a methodology for the design, rigorous documentation, and analysis message-passing applications. We firstly introduce our methodology and describe the model-transformations it entails. An original aspect of our approach is the combination of two models conceived to tackle different facets of message-passing applications. More precisely we rely on *global choreographies* (g-choreographies, for short; see e.g., [13,33] and references therein) to specify the communication pattern of a message-passing system and on *klaimographies* [5] to capture the data-flow and the execution model of our application domain.

We aim to show how a model-driven approach can be conducive of a fruitful collaboration between academics and practitioners. We draw some considerations about this in Section 6. Our approach consists of the following steps:

1. Device a graphical model $G$ representing the coordination among the components of the application; for this we use *global choreographies* (cf. Section 2.1).
2. Transform $G$ into behavioural types formalising the protocol into a behavioural type `K` representing the global behaviour of the application; for this we use *klaimographies* (cf. Section 2.2)

3. Transform K into specifications of each component of the application; for this we project K on *local types* (cf. Section 4).
4. Transform the local types into state machines from which to derive monitors to check for possible deviations from expected behaviour and verify implementations of components (cf. Section 5).

Although, g-choreographies are crucial to settle a common ground between academics and practitioners, they do not capture the data-flow and the execution model of OpenDXL. To cope with this drawback we formalise TIE with *klaimographies*, a data-driven model of choreographies.

*Structure of the paper* An overview of the TIE and an informal account of our behavioural types system is given in Section 2.1 (we refer the reader to [5] for the full details). The behavioural types of TIE are reported in Section 3; there we clarify that our model falls in the setting of "top-down" choreographic approaches. This amounts to say that we first give global specification that formally captures the main aspects of the communication protocol of all TIE from a holistic point of view. Then, in Section 4 we discuss how to automatically derive (by *projection*) the local behaviour of each component of TIE. We consider a few real scenarios in Section 5 and draw some conclusions in Section 6.

## 2 Preliminaries

We survey the two main ingredients of this paper, OpenDXL and klaimographies. We focus on the part of OpenDXL relevant to our case study and only give an informal account of klaimographies (see [5] for details).

### 2.1 An informal account of OpenDXL

The Open Data Exchange Layer (OpenDXL, `https://www.opendxl.com/`) is an open-source initiative aiming to support the exchange of timely and accurate cyber-security information in order to foster the dynamic adaptation of interconnected services to security threats. OpenDXL is part of the McAfee Security Innovation Initiative [23], a consortium of about hundred ICT companies including HP, IBM, and Panasonic.

A main goal of OpenDXL is to provide a shared platform to enable the distributed coordination of security-related operations. This goal is supported by the *threat intelligence exchange* (TIE) reputation APIs [24] designed to enable the coordination of activities involving

- the assessment of the security threats of an environment (configuration files, certificates, unsigned or unknown files, etc.);
- the prioritisation of analysis steps (focusing on malicious or unknown files);
- the customisation of security queries based on reputation-based data (such as product or company names);
- the reaction to suspicious indicators.

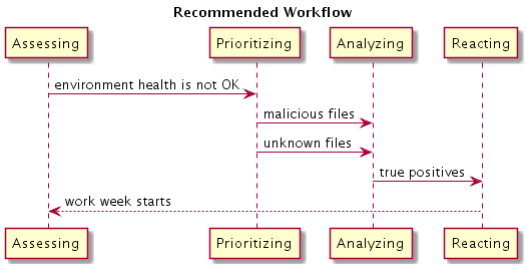


**Fig. 1.** Documenting TIE [24]

A key aspect of OpenDXL lays in its service-oriented nature. Providers use the APIs to offer various services such as reporting services, firewalls, security analytics, etc. Consumers of these APIs (typically companies or large institutions) can either use existing services, or combine them to develop their own functionalities. The basic communication infrastructure features an event-notification architecture whereby participants subscribe to topics of interests to generate events or query services. Such topics are also used to broadcast security information of general interest. The main components of OpenDXL are clients ($\mathsf{C}$), servers ($\mathsf{S}$), and brokers ($\mathsf{B}$). The latter mediate interactions among clients and servers in order to guarantee service availability. Brokers interact with each other to dynamically assign servers to clients when default servers are unavailable.

The high-level workflow of the TIE APIs is specified by the sequence diagram in Fig. 1 (borrowed from [24]). Together with other informal documentation, the diagram guides the implementation of new components or the composition of services available in the platform. For instance, the documentation describing how clients can set the reputation of a file specifies that a client "must have permission to send messages to the `/mcafee/service/tie/reputation/set` topic".

### 2.2 Data-driven global types

Unlike "standard" behavioural types, klaimographies model data flows in a communication model not based on point-to-point interactions. Interactions in a klaimography happen through *tuple spaces* in the style of Linda-like languages [12]. Instead of relying on primitives for sending and receiving messages over a channel, here there are primitives for inserting a tuple on a tuple space, for reading (without consuming) a tuple from a tuple space, or for retrieving a tuple from a tuple space. We call these interactions data-driven, as the coordination is based on (the type of) the exchanged tuples and the *roles* played by components. In fact, the communication model uses pattern matching to establish when a message from a sender may be accessed by a receiver. Crucially, klaimographies also feature *multi-roles*, namely roles that may be enacted by an arbitrary number of instances. Let us discuss these points with a simple example:

$$\mathtt{K} = \mathsf{C} \to \mathsf{S} : (\mathtt{bool} \cdot \mathtt{int})\,@\,\ell\,.\,\mathsf{S} \to \mathsf{C} : (\mathtt{int} \cdot \mathtt{str})\,@\,\ell$$

The klaimography $\mathtt{K}$ specifies the communication protocol between (arbitrarily many) clients $\mathsf{C}$ and (arbitrarily many) servers $\mathsf{S}$. More precisely, each client makes a request to a server by inserting a tuple consisting of a boolean and an integer at the tuple space $\ell$, as indicated by the prefix $\mathsf{C} \to \mathsf{S} : (\mathtt{bool} \cdot \mathtt{int}) \,@\, \ell$. A server consumes the request and generates a response to be consumed by a client, as specified by $\mathsf{S} \to \mathsf{C} : (\mathtt{int} \cdot \mathtt{str}) \,@\, \ell$. Remarkably, $\mathtt{K}$ does not prescribe that the particular client and server involved in the first interaction are also the ones involved in the second interaction; $\mathtt{K}$ above establishes instead that every client starts by producing a tuple to be consumed by a server and then consumes a tuple generated by a server (also $\mathtt{K}$ stipulates that servers behave dually). As a consequence, the participants in $\mathtt{K}$ cannot correlate messages in different interactions. This can be achieved by using binders, e.g.,

$$\mathtt{K}' = \mathsf{C} \to \mathsf{S} : (\mathtt{bool} \cdot \nu \mathtt{x} : \mathtt{int}) \,@\, \ell \,.\, \mathsf{S} \to \mathsf{C} : (\mathtt{x} : \mathtt{int} \cdot \mathtt{str}) \,@\, \ell$$

The first interaction in $\mathtt{K}'$ introduces a new name $\mathtt{x}$ for the integer value exchanged in the first message. The use of $\mathtt{x}$ in the second interaction constraints the instances of $\mathsf{S}$ and $\mathsf{C}$ to share a tuple whose integer expression matches the integer shared in the first interaction. Consequently, the two messages in the protocol are correlated by the integer values in the two messages.

Tuple spaces may simulate other communication paradigms such as multicast or event-notification. For instance, a tuple space $\ell$ can be thought of as a topic; messages can be produced, read and consumed only by those roles that know such topic. Binders can also be used to ensure the creation of new topics. Consider the klaimography below:

$$\mathtt{K}'' = \mathsf{C} \to \mathsf{S} : (\mathtt{bool} \cdot \mathtt{int} \cdot \nu \ell' : \mathtt{loc}) \,@\, \ell \,.\, \mathsf{S} \to \mathsf{C} : (\mathtt{int} \cdot \mathtt{str}) \,@\, \ell'$$

$\mathtt{K}''$ is similar to $\mathtt{K}$ but for the fact that each client communicates to the server a new tuple space $\ell'$ known only to the particular client and server that communicate in the first interaction; the second interaction takes place by producing and consuming messages on such new tuple space.

Broadcast can be achieved by producing persistent messages, e.g.,

$$\mathtt{K}''' = \mathsf{C} \to \mathsf{S} : (\mathtt{bool} \cdot \mathtt{int}) \,@\, \ell \,.\, \mathsf{S}!\mathtt{int} \cdot \mathtt{str} \,@\, \mathtt{r}$$

where $\mathsf{S}!\mathtt{int} \cdot \mathtt{str} \,@\, \mathtt{r}$ states that servers insert their responses at locality $\mathtt{r}$. The absence of round brackets around the tuple expresses that such tuple is read-only (i.e., they cannot be removed from the tuple space); the absence of a receiver expresses that any role can read the tuple; consequently, the generated tuple can be read by any role "knowing" the locality $\mathtt{r}$.

Additionally, klaimographies provide operators for sequential composition ($\prec$), choices ($+$) and recursion ($\mu_\rho \, X.\mathtt{K}$), illustrated in the following section.

## 3 Klaimographies for OpenDXL

The first problem we had to face in the modelling of the protocol was to find a common ground between academic and industrial partners. This is important in

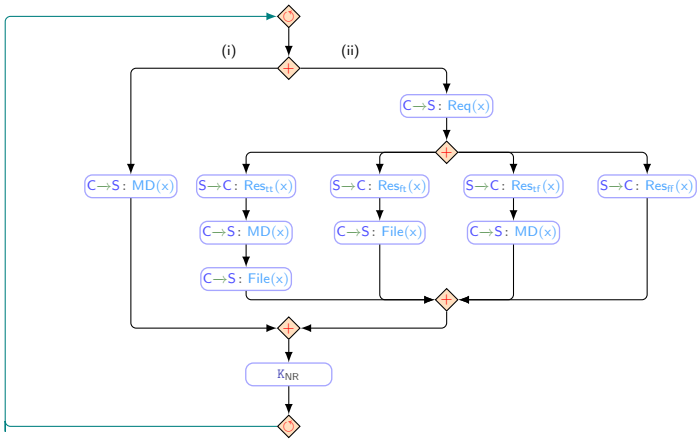


**Fig. 2.** A g-choreography for TIE APIs

order to have enough confidence that the produced formalisation faithfully represent the protocol. To attain this we gave a first approximation of the protocol as the g-choreography in Fig. 2 which we now describe. A client $\mathsf{C}$ and a server $\mathsf{S}$ engage in a protocol where $\mathsf{C}$ may (repeatedly) either (i) send $\mathsf{S}$ meta-data regarding some file $\mathsf{x}$ or (ii) request the analysis of a file $\mathsf{x}$. A server $\mathsf{S}$ reacts to a request from a client in four possible ways depending on the information $\mathsf{S}$ may need to further acquire from the requesting client. In the protocol these alternatives are encoded with a message $\mathsf{Res}_{\mathsf{bb'}}(\mathsf{x})$ where $\mathsf{b}$ and $\mathsf{b'}$ are two boolean flags; the first boolean is set to true when the server needs meta-data related to the file $\mathsf{x}$ while $\mathsf{b'}$ is set to true if more context information about the file is necessary. The client reacts to this request from the sever as appropriate. For instance, if $\mathsf{C}$ receives the message $\mathsf{Res}_{\mathsf{tt}}(\mathsf{x})$ then it has to send both meta-data and context information, while only the latter are sent if $\mathsf{Res}_{\mathsf{ft}}(\mathsf{x})$ is received. Before iterating back, the server may publish a new report[4]; this is modelled by the activity $\mathtt{K}_{\mathrm{NR}}$ which we leave unspecified. This activity consists of a possible emission of a new report about file $\mathsf{x}$ that the server $\mathsf{S}$ may decide to multi-cast to clients (not just to clients currently engaging with the server).

We remark that the g-choreography in Fig. 2 represents the interactions between clients and servers and has been introduced as a first step in the formalisation of the protocol to pave the way for its algebraic definition as klaimographies. Firstly, a graphical representation played a central rôle when validating protocol interactions with industrial partners. Secondly, the graph was used as a blueprint for the formalisation. Hence, we invite the reader to follow such graph as the formal definitions unroll.

In the OpenDXL platform several clients and servers may interact by exchanging messages. The interaction in TIE is always triggered by a client which, as seen in Section 2.1, iteratively decides to either send some metadata on a file

[4] The server is actually multi-threaded and could issue new reports about files other than $\mathsf{x}$ at any time; for simplicity, we do not model this aspect.

or request for the reputation of a specific file. This can be defined as follows

$$\mathtt{K}_{\text{TIE}} \triangleq \mu_{\mathtt{C}}\, X.\mathtt{K}_{\text{BODY}} \prec X \tag{1}$$

where $\mu_{\mathtt{C}}\, X.\mathtt{K}_{\text{BODY}} \prec X$ is the recursive type to express iterative behaviour; it indicates that role $\mathtt{C}$ is the one controlling the iteration. Namely, $\mathtt{C}$ decides whether to repeat the execution of the body $\mathtt{K}_{\text{BODY}}$ or to end it. The sequential composition $\mathtt{K}_{\text{BODY}} \prec X$ is just syntax to express that, after the execution of $\mathtt{K}_{\text{BODY}}$, the iteration restarts.

**Notation** *We write* $\_ \triangleq \_$ *as "macros" so that occurrences of the left-hand side of the equation are verbatim replaced for its right-hand side.*

The body of the iteration in (1), defined as

$$\mathtt{K}_{\text{BODY}} \triangleq (\mathtt{K}_{\text{MD}}(\mathtt{x},\ell) + \mathtt{K}_{\text{REQ}}(\mathtt{x},\ell)) \prec \mathtt{K}_{\text{NR}}(\mathtt{x}) \tag{2}$$

specifies that each iteration consists of a choice between $\mathtt{K}_{\text{MD}}(\mathtt{x},\ell)$ and $\mathtt{K}_{\text{REQ}}(\mathtt{x},\ell)$ followed by $\mathtt{K}_{\text{NR}}(\mathtt{x})$:

- The branch $\mathtt{K}_{\text{MD}}(\mathtt{x},\ell)$ accounts for the case in which a client sends new metadata to a server.
- The branch $\mathtt{K}_{\text{REQ}}(\mathtt{x},\ell)$ describes the interaction for the case in which the client sends a reputation request.
- The continuation $\mathtt{K}_{\text{NR}}(\mathtt{x})$ describes the decision of the server of emitting a reputation report.

**Notation** *In accordance with the previous notation,* $\mathtt{x}$ *and* $\ell$ *above are just meta-identifiers for the same syntactic identifier across equations.*

Let $\mathtt{b}$ be a globally known location representing the public name on which a client sends requests to a server. The branches of the body are defined as:

$$\begin{aligned}
\mathtt{K}_{\text{MD}}(\mathtt{x},\ell) &\triangleq \mathtt{C} \to \mathtt{S} : (\mathtt{MD} \cdot \nu \mathtt{x} : \mathtt{Dgt} \cdot \nu \ell : \mathtt{loc})\,@\,\mathtt{b} \\
\mathtt{K}_{\text{REQ}}(\mathtt{x},\ell) &\triangleq \mathtt{C} \to \mathtt{S} : (\mathtt{Req} \cdot \nu \mathtt{x} : \mathtt{Dgt} \cdot \nu \ell : \mathtt{loc})\,@\,\mathtt{b}\,.\,\mathtt{K}_{\text{INFO}}(\mathtt{x},\ell)
\end{aligned} \tag{3}$$

In both cases the first interaction takes place on the tuple space $\mathtt{b}$.

In $\mathtt{K}_{\text{MD}}(\mathtt{x},\ell)$, the client simply sends a tuple $\mathtt{MD} \cdot \nu \mathtt{x} : \mathtt{Dgt} \cdot \nu \ell : \mathtt{loc}$ made of three fields. The first field has sort $\mathtt{MD}$ which is a tag for messages carrying metadata. The second field is a *named* sort $\nu \mathtt{x} : \mathtt{Dgt}$, where (i) the sort $\mathtt{Dgt}$ (after digest) types values that are hash codes of files and (ii) the identifier $\mathtt{x}$ is introduced to establish the correlation that will be used in the following interactions. This mechanism enables the tracking of data dependencies among interactions. Finally, the third field is another named sort $\ell : \mathtt{loc}$; basically, the client communicates also the name $\ell$ of a new tuple space, to be used in the subsequent communications. For instance, the continuation type

$$\mathtt{K}_{\text{NR}}(\mathtt{x}) \triangleq \quad \mathtt{S!Report} \cdot \mathtt{x} : \mathtt{Dgt}\,@\,\mathtt{b'} + \mathbf{0}$$

describes the behaviour of a server that decides whether to emit a new report about the received metadata or not. Type $\mathtt{K}_{\text{NR}}(\mathtt{x})$ consists of a non-deterministic

choice between a branch $\mathsf{S}!\mathtt{Report}\cdot\mathtt{x}:\mathtt{Dgt}\,@\,\mathtt{b'}$ and the empty type $\mathbf{0}$. The former specifies that the server publishes a new report for the file by emitting a (persistent) tuple of type $\mathtt{Report}\cdot\mathtt{x}:\mathtt{Dgt}$ on a publicly known[5] tuple space $\mathtt{b'}$. Note that the use of $\mathtt{x}$ constraints the new report produced by server $\mathsf{S}$ to be related to a file digest communicated earlier to $\mathsf{S}$.

The interaction prefixes $\mathsf{C}\to\mathsf{S}:(\ldots)\,@\,\ell$ are quite different than the prefix $\mathsf{S}!\mathtt{Report}\cdot\mathtt{x}:\mathtt{Dgt}\,@\,\ell$. This is a remarkable peculiarity of klaimographies that is quite useful to model TIE. Firstly, the former kind of prefix describes an interaction between two roles: clients are supposed to produce messages of some sort for servers. Instead, the behavioural type $\mathsf{S}!\mathtt{Report}\cdot\mathtt{x}:\mathtt{Dgt}\,@\,\ell$ only prescribes the expected communication from a single role, the server. This allows *any* role to access the tuple types generated by this kind of prefixes.

Another important aspect is the other syntactic difference: the messages in round brackets are produced to be consumed, while the ones not surrounded by brackets are persistent and can only be read; moreover, the message can be read by any role able to access the tuple space $\ell$. For instance, requests of clients are eventually handled by a server, while any role can read, but not remove, reports.

Let us now return to the comment on the other branch in (2). In the klaimography $\mathtt{K}_{\text{REQ}}(\mathtt{x},\ell)$, a client sends a request for the reputation of a file by sending a message whose tag is of type $\mathtt{Req}$. In that message, the client sends the digest $\mathtt{Dgt}$ that identifies the file and, analogously to $\mathtt{K}_{\text{MD}}(\mathtt{x},\ell)$, a fresh locality $\ell$; the correlation $\mathtt{x}$ and the locality $\ell$ are used in the subsequent interactions, which are described by $\mathtt{K}_{\text{INFO}}(\mathtt{x},\ell)$ below.

$$\begin{aligned}\mathtt{K}_{\text{INFO}}(\mathtt{x},\ell) \triangleq\ & \mathsf{S}\to\mathsf{C}:(\mathtt{Res}_{\mathtt{tt}}\cdot\mathtt{x}:\mathtt{Dgt})\,@\,\ell\,.\,\mathtt{K}_{\text{TT}}(\mathtt{x},\ell)\\ &+\mathsf{S}\to\mathsf{C}:(\mathtt{Res}_{\mathtt{tf}}\cdot\mathtt{x}:\mathtt{Dgt})\,@\,\ell\,.\,\mathtt{K}_{\text{TF}}(\mathtt{x},\ell)\\ &+\mathsf{S}\to\mathsf{C}:(\mathtt{Res}_{\mathtt{ft}}\cdot\mathtt{x}:\mathtt{Dgt})\,@\,\ell\,.\,\mathtt{K}_{\text{FT}}(\mathtt{x},\ell)\\ &+\mathsf{S}\to\mathsf{C}:(\mathtt{Res}_{\mathtt{ff}}\cdot\mathtt{x}:\mathtt{Dgt})\,@\,\ell\end{aligned}$$

This klaimography corresponds to the inner-most choice of the graph in Section 2.1; it prescribes the possible responses that the server may send to the client. We start commenting on the last branch. If the server does not require further information, it simply informs the client that the interaction for that request concludes. The remaining branches of $\mathtt{K}_{\text{INFO}}(\mathtt{x},\ell)$ model the cases in which the server requests both the metadata and the file (first branch), just the metadata (second branch) or just the file (third branch). When both metadata and file are requested, then the protocol continues as follows

$$\mathtt{K}_{\text{TT}}(\mathtt{x},\ell) \triangleq \mathsf{C}\to\mathsf{S}:(\mathtt{MD}\cdot\mathtt{x}:\mathtt{Dgt})\,@\,\ell\,.\,\mathsf{C}\to\mathsf{S}:(\mathtt{File}\cdot\mathtt{x}:\mathtt{Dgt})\,@\,\ell$$

And, when the server asks for either the metadata or the file, then

$$\begin{aligned}\mathtt{K}_{\text{TF}}(\mathtt{x},\ell) &\triangleq \mathsf{C}\to\mathsf{S}:(\mathtt{MD}\cdot\mathtt{x}:\mathtt{Dgt})\,@\,\ell\\ \mathtt{K}_{\text{FT}}(\mathtt{x},\ell) &\triangleq \mathsf{C}\to\mathsf{S}:(\mathtt{File}\cdot\mathtt{x}:\mathtt{Dgt})\,@\,\ell\end{aligned}$$

[5] Here we simplify the actual implementation where the topic used to publish the report is related to the file used in the request.

which is in accordance with the g-choreography in Section 2.1.

## 4 Projections

As commonplace in choreographic approaches, the description of the expected behaviour of each participant in a protocol can be obtained by *projection*. In our case, this is an operation that takes a klaimography and a role and generates a description, dubbed *local* type, of the flow of messages sent and received by that participant. Local types are meant to give an abstract specification of the processes implementing the roles of the klaimography. We write the projection of a klaimography $\mathtt{K}$ for the role $\rho$ as $\mathtt{K}\restriction_{\rho}$. Note that the projection operation is completely automatic; given a klaimography the behaviour of each component is algorithmically derived. We omit here the formal definition of $\mathtt{K}\restriction_{\rho}$, which can be found at [5], and illustrate its application to $\mathtt{K}_{\text{TIE}}$ in (1).

We consider $\mathtt{K}_{\text{TIE}}\restriction_{\mathsf{C}}$ first. The projection operation is defined by induction on the syntax of the klaimography; hence we focus on the constituent parts of $\mathtt{K}_{\text{TIE}}$. Consider the branch $\mathtt{K}_{\text{MD}}(\mathtt{x},\ell)$, which is defined in (3) as the interaction $\mathsf{C}\to\mathsf{S}:(\mathtt{MD}\cdot\nu\mathtt{x}:\mathtt{Dgt}\cdot\nu\ell:\mathtt{loc})@\mathtt{b}$. The projection of this interaction on the client role just consists of the behaviour that generates a message of type $\mathtt{MD}\cdot\nu\mathtt{x}:\mathtt{Dgt}\cdot\nu\ell:\mathtt{loc}$ on the locality $\mathtt{b}$; formally, this is written

$$\mathtt{K}_{\text{MD}}(\mathtt{x},\ell)\restriction_{\mathsf{C}} = (\mathtt{MD}\cdot\nu\mathtt{x}:\mathtt{Dgt}\cdot\nu\ell:\mathtt{loc})!\mathtt{b}$$

Note (a) the use of the round brackets to represent message consumption, and (b) the projection is oblivious of the intended receiver (the server). In fact, the behavioural type system of klaimographies ensures that if the actual components abide by the klaimographies given in Section 3, then only components enacting the role of the server will access those kind of tuples.

The projection for $\mathtt{K}_{\text{REQ}}(\mathtt{x},\ell)$ (and all its constituents) is analogous:

$$\mathtt{K}_{\text{REQ}}(\mathtt{x},\ell)\restriction_{\mathsf{C}} = (\mathtt{Req}\cdot\nu\mathtt{x}:\mathtt{Dgt}\cdot\nu\ell:\mathtt{loc})!\mathtt{b}\,.\,\mathtt{K}_{\text{INFO}}(\mathtt{x},\ell)\restriction_{\mathsf{C}}$$

$$\begin{aligned}\mathtt{K}_{\text{INFO}}(\mathtt{x},\ell)\restriction_{\mathsf{C}} = {} & (\mathtt{Res}_{\mathtt{tt}}\cdot\mathtt{x}:\mathtt{Dgt})?\ell\,.\,\mathtt{K}_{\text{TT}}(\mathtt{x},\ell)\restriction_{\mathsf{C}} \\ & + (\mathtt{Res}_{\mathtt{tf}}\cdot\mathtt{x}:\mathtt{Dgt})?\ell\,.\,\mathtt{K}_{\text{TF}}(\mathtt{x},\ell)\restriction_{\mathsf{C}} \\ & + (\mathtt{Res}_{\mathtt{ft}}\cdot\mathtt{x}:\mathtt{Dgt})?\ell\,.\,\mathtt{K}_{\text{FT}}(\mathtt{x},\ell)\restriction_{\mathsf{C}} \\ & + (\mathtt{Res}_{\mathtt{ff}}\cdot\mathtt{x}:\mathtt{Dgt})?\ell\end{aligned}$$

$$\mathtt{K}_{\text{TT}}(\mathtt{x},\ell)\restriction_{\mathsf{C}} = (\mathtt{MD}\cdot\mathtt{x}:\mathtt{Dgt})!\ell\,.(\mathtt{File}\cdot\mathtt{x}:\mathtt{Dgt})!\ell$$

$$\mathtt{K}_{\text{TF}}(\mathtt{x},\ell)\restriction_{\mathsf{C}} = (\mathtt{MD}\cdot\mathtt{x}:\mathtt{Dgt})!\ell$$

$$\mathtt{K}_{\text{FT}}(\mathtt{x},\ell)\restriction_{\mathsf{C}} = (\mathtt{File}\cdot\mathtt{x}:\mathtt{Dgt})!\ell$$

Observe that the projection for $\mathtt{K}_{\text{INFO}}$ is a choice in which $\mathsf{C}$ expects (and consumes) one of the four possible messages produced by the server at locality $\ell$.

Finally, the projection of $\mathtt{K}_{\text{NR}}(\mathtt{x})$ is

$$\mathtt{K}_{\text{NR}}(\mathtt{x})\restriction_{\mathsf{C}} = \mathtt{Report}\cdot\mathtt{x}:\mathtt{Dgt}?\mathtt{b'} + \mathbf{0}$$

Differently from the projection of interactions in which the client consumes the messages, the first branch of the above projection just reads the message at the locality $\ell$. Note the difference between $(\mathtt{t})?\ell$ (consumption) and $\mathtt{t}?\ell$ (read), which reflects the usage of round bracket discussed in Section 3.

Projection works homomorphically on choices and sequential composition, hence the projection of $\mathtt{K}_{\text{BODY}}$ in (2) we have

$$\mathtt{K}_{\text{BODY}} \upharpoonright_{\mathsf{C}} = (\mathtt{K}_{\text{MD}}(\mathtt{x},\ell) \upharpoonright_{\mathsf{C}} + \mathtt{K}_{\text{REQ}}(\mathtt{x},\ell) \upharpoonright_{\mathsf{C}}) \prec \mathtt{K}_{\text{NR}}(\mathtt{x}) \upharpoonright_{\mathsf{C}}$$

We now give the projection of $\mathtt{K}_{\text{TIE}}$, which is a recursive klaimography. Then,

$$\mathtt{K}_{\text{TIE}} \upharpoonright_{\mathsf{C}} = \big(\mu X(\mathtt{b}) . \langle \mathtt{stop} \rangle ! \mathtt{b} . \mathbf{0} + \langle \nu \mathtt{y} : \mathtt{loc} \rangle ! \mathtt{b} . \mathtt{K}_{\text{BODY}} \upharpoonright_{\mathsf{C}} \prec X\langle y \rangle \big) \langle \mathtt{b} \rangle \tag{4}$$

The projection of a recursive klaimography is also a recursive local type. However, the projection introduces auxiliary interactions to coordinate the execution of the loop. Since $\mathsf{C}$ is the role that coordinates the recursion in $\mathtt{K}_{\text{TIE}}$, in the projection $\mathsf{C}$ starts its body by communicating its decision to terminate or to continue. Namely, the body of $\mathtt{K}_{\text{TIE}} \upharpoonright_{\mathsf{C}}$ has two branches, $\langle \mathtt{stop} \rangle ! \mathtt{b}$ communicates the termination of the recursion, while the other starting with $\langle \nu \mathtt{y} : \mathtt{loc} \rangle ! \mathtt{b}$ iterates (and distributes a fresh localities for the next iteration).

Note that recursive variables $X$ in the local types are parameterised variables $X(\mathtt{b})$ and $X\langle \mathtt{b} \rangle$. In general, a klaimography $\mu_\rho \; X.\mathtt{K}$ is projected as a recursive local type $\big(\mu X(\widetilde{x}) \,.\, \mathtt{L}\big)\langle \widetilde{\ell} \rangle$ where the formal parameters $\widetilde{x}$ stand for the locations used for coordination and $\widetilde{\ell}$ are the initial values, in this case, $\mathtt{b}$. The projection for the behaviour of the server is obtained analogously.

## 5 Types at work

Like data types, behavioural types can be regarded as specifications of the intended behaviour of a system. As such they can check that the components implementing the protocol abide by their specifications. Customarily, approaches to behavioural types focus on static enforcement [15,9,17], i.e., the source code implementing a role is type-checked against its local type and the soundness of the type checking algorithm ensures that well-typed code behaves as prescribed by its type. Also the dynamic enforcement of protocols based on local types has been addressed in the literature [3,28,11]. In most cases, monitors dynamically check that the messages exchanged by the components comply with the protocol. Deviations from the expected behaviour are singled out and offending components are blamed.

In this work we explore the usage of local types for the off-line monitoring of role implementations. In particular, we use projections to check that the different implementations of the multirole $\mathsf{C}$ in TIE follow the protocol. We take advantage of the fact that the communication infrastructure of TIE keeps a log with the communication messages generated by the different roles.

In Fig. 3 we show an anonymised (and simplified) version of a few entries of a real log. Each entry corresponds to an interaction between a client and a server and it consists of a record of comma-separated fields which we now describe:

```
2019-03-27T15:59:49, 649, clientA, server1, Req, file1
2019-03-27T15:59:49, 649, server1, clientA, Res, 1, 0, file1
2019-03-27T15:59:50, 649, clientA, server1, MD, file1
2019-03-27T15:59:50, 340, clientC, server1, Req, file2
2019-03-27T15:59:50, 340, server1, clientC, Res, 1, 1, file2
2019-03-27T15:59:50, 699, clientD, server1, MD, file2
2019-03-27T15:59:50, 340, clientC, server1, File, file2
2019-03-27T15:59:51, 021, clientE, server1, Req, file3
2019-03-27T15:59:51, 021, server1, clientE, Res, 0, 0, file3
2019-03-27T15:59:51, 370, clientF, server1, MD, file3
2019-03-27T15:59:51, 721, server1, broadcast, Report, file3
...
```

**Fig. 3.** A simplified snippet of a real (anonymised) log

- the first field is a global timestamp used to order the entries chronologically;
- the second field is the *locality*, which is encoded by a three-digits number;
- the third and fourth fields are the identity of the *sender* and of the *receiver* respectively (for obvious reasons, the real identities have been obfuscated; Fig. 3 uses symbolic names `clientA`, `server1`, etc.);
- the remaining fields are the payloads of the message, which varies depending on the type of the message.

The type of each message is identified by a tag: `Req`, `MD`, and `File` have analogous meaning to the ones used in the specification of the protocol in Sections 3 and 4. The sorts such as $\mathtt{Res}_{\mathtt{tf}}$ used in our specification are rendered in the implementation with a payload consisting of three parts: the tag `Res` and two binary digits; used to encode the subscript (with `1` representing `true` and `0` representing `false`); for instance, the subscript `tf` above is encoded as the pair `1`, `0`. We use $\mathtt{file}_i$ to represent the different digests transmitted over the messages.

The first entry in the log of Fig. 3 is generated by the interaction

$$\mathsf{C} \to \mathsf{S} : (\mathtt{Req} \cdot \nu \mathtt{x} : \mathtt{Dgt} \cdot \nu \ell : \mathtt{loc}) \mathbin{@} \mathtt{b}$$

executed by $\mathsf{K}_{\text{REQ}}(\mathtt{x}, \ell)$, where the instance `clientA` of the role $\mathsf{C}$ sends to the instance `server1` of $\mathsf{S}$ a request for a reputation report about the file `file1`. The second entry in the log corresponds to the selection of the branch

$$\mathsf{S} \to \mathsf{C} : (\mathtt{Res}_{\mathtt{tf}} \cdot \mathtt{x} : \mathtt{Dgt}) \mathbin{@} \ell \,.\, \mathsf{K}_{\text{TF}}(\mathtt{x}, \ell)$$

in $\mathsf{K}_{\text{INFO}}(\mathtt{x}, \ell)$ in which the server asks the client for the metadata of the file; the messages in which the client sends the metadata can be seen in the third line of the log. Obviously, the interactions among different instances need not to be consecutive, as it is the case for the entries at locality `340` which are on the lines `4`, `5` and `7`. Observe also that the last entry in Fig. 3 has `broadcast` as its receiver. This message corresponds to the publication of a reputation report by the server, which is defined in $\mathsf{K}_{\text{NR}}(\mathtt{x})$ as $\mathtt{Report} \cdot \mathtt{x} : \mathtt{Dgt}?\ell$.

We have implemented in `Python` an off-line monitor that takes a log and a local type in input and checks whether the log faithfully follows behaviour described by the local type. Local types are turned into a textual representation of finite state automata that can be depicted as UML state machines. For instance,

```
@startuml left to right direction
[*] --> S0
S0 --> S0: 'MD'@l @Dgt
S0 --> S1: 'Req'@l
S1 --> S2: ('Res', '1', '1')@l -> @Dgt
S1 --> S3: ('Res', '0', '1')@l -> @Dgt
S1 --> S4: ('Res', '1', '0')@l -> @Dgt
S1 --> S0: ('Res', '0', '0')@l
S2 --> S3: 'MD'@l -> @Dgt
S3 --> S0: 'File'@f
S4 --> S0: 'MD'@l
@enduml
```

**Fig. 4.** $\mathtt{K}_{\text{DXL}} \upharpoonright_{\mathtt{C}}$ as UML diagram (textual representation)

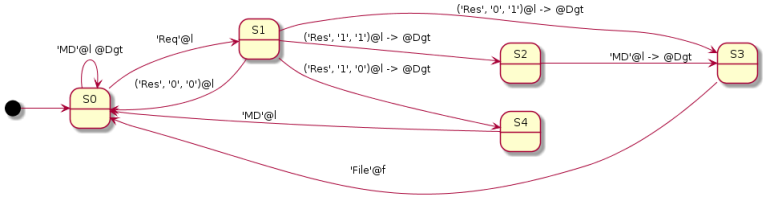


**Fig. 5.** $\mathtt{K}_{\text{DXL}} \upharpoonright_{\mathtt{C}}$ as UML diagram (graphical representation)

the local type $\mathtt{K}_{\text{TIE}} \upharpoonright_{\mathtt{C}}$ is defined as shown in Fig. 4, which can be graphically represented as shown in Fig. 5.

These representations are obtained by "massaging" the projections defined in Section 4. The main difference between the UML representation and the local type (besides the obvious syntactic changes) is that the former does not contain the messages for coordinating the recursion in (4) (i.e., $\mathtt{stop}$ and $\nu \mathtt{y} : \mathtt{loc}$); those have been omitted because not explicitly exchanged by the components. As a consequence, we assume that the client continues the loop if it keeps sending messages and it finishes silently otherwise. Another simplification for the sake of the presentation is the omission of $\mathtt{K}_{\text{NR}}(\mathtt{x}) \upharpoonright_{\mathtt{C}}$, essentially because the observable behaviour of the client is unaffected if it reads or not a report. In fact, the log is not informative enough to discriminate on the choice made by the client.

Once such simplifications are in place, (4) can be easily matched with the graphical representation in Fig. 5. The state $\mathtt{S0}$ represents $\mathtt{K}_{\text{TIE}} \upharpoonright_{\mathtt{C}}$. The self-loop stands for the selection of the branch $\mathtt{K}_{\text{MD}}(\mathtt{x}, \ell) \upharpoonright_{\mathtt{C}}$, i.e., the client sends a message containing metadata, and then restart the loop. The transition from $\mathtt{S0}$ to $\mathtt{S1}$ represents instead the choice of the branch $\mathtt{K}_{\text{REQ}}(\mathtt{x}, \ell) \upharpoonright_{\mathtt{C}}$, i.e., the client request of a reputation report. The remaining states are in one-to-one correspondence with the following projections defined in the previous section: $\mathtt{S1}$ stands for $\mathtt{K}_{\text{INFO}}(\mathtt{x}, \ell) \upharpoonright_{\mathtt{C}}$, $\mathtt{S2}$ for $\mathtt{K}_{\text{TT}}(\mathtt{x}, \ell) \upharpoonright_{\mathtt{C}}$, $\mathtt{S3}$ for $\mathtt{K}_{\text{FT}}(\mathtt{x}, \ell) \upharpoonright_{\mathtt{C}}$, and $\mathtt{S4}$ for $\mathtt{K}_{\text{TF}}(\mathtt{x}, \ell) \upharpoonright_{\mathtt{C}}$. All the transitions are decorated with the associated messages sent or received by a client. Note also that $\mathtt{S1}$, $\mathtt{S3}$ and $\mathtt{S4}$ have transitions to the state $\mathtt{S0}$ meaning that execution of the body the is completed and that the body can be restarted.

With this implementation we have detected a few deviations from the expected behaviour. In particular, some clients exhibit the following violations:

– files are sent for analysis without a prior request,
– requests for further information from the server are not honoured.

The first violation is detected by the presence of an entry of the log with a message tagged `File` without a previous message from the server with tag $\texttt{Res}_{\texttt{tt}}$ or $\texttt{Res}_{\texttt{ft}}$. The second violation is due to the absence of an entry related to a given hash used by the server for asking further information.

Our implementation can also check other properties. For example, TIE clients should guarantee a so-called "time-window" property which requires that

> "a request for the analysis of the same file from a client must not happen before a given amount of time elapsed from the previous request from the client for the same file."

This property (as well as others) can be checked by monitor derived from the local types as done in the examples above.

## 6 Conclusions, Related & Future Work

*Summary* We reported on a collaboration between industrial and academic partners which applied formal methods to address a key problem affecting APIs-based software. More precisely, the problem that informal specifications of the behaviour of services may lead to errors in message-passing applications. For instance, third-party clients of TIE services exhibit anomalous when interacting with the services developed at McAfee. To overcome this problem, TIE services are engineered with a rather defensive approach to anticipate anomalous interactions. Unintended behaviours are reported to third-parties after a "post-mortem" analysis of execution logs.

We devised a model-driven approach to model and validate message-passing software. We applied the methodology in the context of the OpenDXL platform, an initiative of a consortium of industries conceived for the development of cyber-security functionalities. The platform provides an API to allow developers to access and combine the functionalities of a service-oriented architecture. In this context we applied the methodology to the *threat intelligence exchange* (TIE) service provided by McAfee Cordoba for the assessment of security threats, prioritisation of analysis steps, reputation-based data queries.

*Related work* The use of behavioural types for the specification and analysis of message-passing application is widespread (see [16] for a survey). Semantics of behavioural types (operational or denotational) abstract the behaviour of systems and enables the use of formal methods and tools to check their properties.

Our proposal hinges on a form of choreographies in the vein of global type systems [15], which formally capture the design of WSCDL [19]. In fact, the specification of a global view is the starting step of our methodology and the use of a projection operation to (automatically) derive local views is a paramount step in the model-transformation chain described in Section 1. The literature

offers several variants of choreographic models [8,31,4,6,33,14] (to mention but a few). A common treat of those models is that they are grounded on point-to-point communication in traditional settings (such as the use of the actor model [1] or $\pi$-calculus [32,26,27]). A distinguished feature of OpenDXL is that it relies on event-notification mechanisms. This is the main motivation for the adoption of *klaimographies* [5]. In fact, unlike other choreographic approaches, klaimographies advocate a peculiar interpretation of interactions. More precisely, interactions $\mathsf{A}\to\mathsf{B}\colon \mathsf{m}$ are generally interpreted as "*an instance* of $\mathsf{A}$ and *an instance* of $\mathsf{B}$ exchange message $\mathsf{m}$". The interpretation of $\mathsf{A}\to\mathsf{B}\colon \mathsf{m}$ drastically changes in klaimographies and becomes "*any instance* of $\mathsf{A}$ generates the message $\mathsf{m}$ expected to be handled by *any instance* of $\mathsf{B}$". This interpretation is the cornerstone for a faithful modelling of OpenDXL.

*Lesson learned* Although we are at an early stage of the collaboration, we can draw some conclusions.

A first point worth remarking is about the **effectiveness** of our methodology. On the one hand, the academic partners were oblivious of several current practices (such as the continuous defensive patching TIE servers). On the other hand, the industrial partners acquired some notions about behavioural specifications during the participation of a school [25] organised by the academic partners as well as presented the OpenDXL platforms at the school. The methodology was applied immediately after the school and the bulk of modelling and analysis of TIE was concluded in about 3-persons month. In the chain of model transformations of our methodology, steps (1) and (4) were paramount for practitioners to apply this methodology: the use of visual, intuitive, yet formal models enabled a fruitful collaboration among stakeholders. In fact, g-choreographies were key to tune up the model and to identify the main aspects of the intended communication protocol as well as to ease the collaboration between practitioners and academics. Basically, g-choreographies gave a first intuitive presentation capturing the essential interactions of TIE. This has been instrumental for an effectual collaboration. Once the g-choreography expressing the intended behaviour has been identified, the academic partners have devised the klaimographies formalising the expected behaviour. The identification of the corresponding klaimographies allowed us to automatically derive local specifications (step (iii)) and use them as precise blue-prints of components as well as to automatically derive monitors (step (iv)). Remarkably, the transformation from local types to state machines was suggested by our industrial partners who saw it as a more streamlined way of sharing the specifications among practitioners (including those outside McAfee). At this stage we do not have data to measure the impact of the enhanced documentation on the quality of the software produced.

This experience also highlights the importance of **non-deterministic abstractions** and of **visual tools** in practice. We argue that these elements are paramount for collaborations that could be beneficial to both academics and practitioners. In fact, behavioural types (as many formal methods) may not be easy for practitioners to handle. To tackle this issue we opted for models offering a visual and intuitive presentations of the formal models used in the specifi-

cations. The specification in terms of g-choreographies and klaimographies was attained in few days of man-power involving academics and practitioners. This hints that our model-driven methodology can significantly reduce the steepness of the learning curve that formal methods often require.

The problem of informal behavioural specification is ubiquitous in API-based software. The approach we followed aimed at some **generality**: instead of devising ad-hoc formal methods for the OpenDXL case study, we decided to apply existing frameworks. In fact, both g-choreographies and klaimographies had been developed before and independently of this collaboration. The methodology proposed here assumes only that components communicate through generative coordination mechanisms [12]. As noted by one of the reviewers, "tuple-semantics are well-suited not only for this use case but for the modern age of IoT, where event-based middlewares are becoming the norm."

A final note on the connection with other formal methods. Behavioural specifications offer also support to "bottom-up" engineering (see, e.g., [20,22]). This would require to infer the behaviour to analyse from logs and, as noted by another reviewer, one could spare "to model the whole behaviour [...] and focus on specific components." We concur that our methodology can be complemented by such technique (and this is indeed one of the goals within the BehAPI project). Also, one may wonder if the methodology can be combined with model checking. This is indeed the case since our models feature operation semantics amenable to be model checked. A drawback of model checking is that practitioners would find it hard to express the properties to check. Instead the top-down approach allowed them to express such conditions in terms of state machines.

*Future work* Global graphs have been key to facilitate the collaboration between academics and industrial partners for the former can use g-choreographies precisely (since they come with a precise semantics) and the latter can use the visual and intuitive presentation of g-choreographies. It is in the scope of future work to use the formal framework of g-choreographies. In fact, we can use g-choreographies to verify liveness properties of the communication protocols, or to generate executable template code to be refined by practitioners. We plan to extend **ChorGram** [21], a tool based on g-choreographies, to support the methodology. For instance, projection operations from global to local views are a key feature of our choreographic framework. Here, we have manually given klaimographies and their projections. This can be automatised by algorithmically transforming g-choreographies into klaimographies. Another possibility is to exploit **ChorGram** to generate code; for instance, **ChorGram** can map g-choreographies to (executable) Erlang code. These sort of functionalities are highly appealing to industrial stakeholders due (a) to the "correct-by-construction" principle they support and (b) to the fact that each release of TIE services requires the realisation of in-house clients for many different languages and platforms. For instance, OpenDXL needs to develop several version of each component for different execution environments. Also, TIE clients have to be implemented in different programming languages or for operating systems; this could be done by devising each software component by projection from a global view. Having tools

that generate template code for implementing the communication protocol of each component would speed up the development process and reduce the time of testing (which would not need to focus on communications which would be correct-by-construction). In order to attain this, it could be useful to "dress up" g-choreographies with existing industrial standards that practitioners may find more familiar (and may be more appealing). An interesting candidate for this endeavour is BPMN [29] since its coordination mechanisms are very close to those of g-choreographies. In fact, BPMN is becoming popular in industry and it has recently gained the attention of the scientific community which is proposing formal semantics of its constructs. For instance, the formal semantics in [7] could be conducive of a formal mapping from BPMN to g-choreographies or global types. In this way practitioners may specify global views within a context without spoiling the rigour of our methodology.

For simplicity in this paper we abstracted away from some aspects of TIE. The extension of our approach to the complete protocol is not conceptually complex, but it is scope for future work. This will include the analysis to further properties expected of TIE components and that can be checked from the logs. Following our methodology, we plan to devise monitors for the run-time verification of those properties as well.

A final remark is about other advantages of behavioural types that we can exploit in the future. For instance, one goal is to device tools for checking the compliance of components to the TIE protocol. This can be achieved by type-checking components against their projections.

*Acknowledgments* We thank the anonymous reviewers for their many insightful comments and suggestions.

## References

1. G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, MA, USA, 1986.
2. W. Ariola and C. Dunlop. Testing in the API Economy. Top 5 Myths. `https://alm.parasoft.com/api-testing-myths`.
3. L. Bocchi, T.-C. Chen, R. Demangeon, K. Honda, and N. Yoshida. Monitoring networks through multiparty session types. In *Formal Techniques for Distributed Systems*, pages 50–65. Springer, 2013.
4. M. Bravetti and G. Zavattaro. Contract compliance and choreography conformance in the presence of message queues. In R. Bruni and K. Wolf, editors, *WS-FM 2008*, LNCS. Springer, Sept. 2008. To appear.
5. R. Bruni, A. Corradini, F. Gadducci, H. C. Melgratti, U. Montanari, and E. Tuosto. Data-driven choreographies à la klaim. In *Models, Languages, and Tools for Concurrent and Distributed Programming - Essays Dedicated to Rocco De Nicola on the Occasion of His 65th Birthday*, pages 170–190, 2019.
6. N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. Choreography and orchestration conformance for system design. In *Coordination Models and Languages, 8th International Conference, COORDINATION 2006, Proceedings*, pages 63–81, 2006.

7. F. Corradini, A. Morichetta, B. Re, and F. Tiezzi. Walking through the semantics of exclusive and event-based gateways in BPMN choreographies. In M. S. Alvim, K. Chatzikokolakis, C. Olarte, and F. Valencia, editors, *The Art of Modelling Computational Systems: A Journey from Logic and Concurrency to Security and Privacy - Essays Dedicated to Catuscia Palamidessi on the Occasion of Her 60th Birthday*, LNCS, pages 163–181. Springer, 2019.
8. M. Dalla Preda, M. Gabbrielli, S. Giallorenzo, I. Lanese, and M. Jacopo. Dynamic choreographies - safe runtime updates of distributed applications. In *COORDINATION 2015*, pages 67–82, 2015.
9. M. Dezani-Ciancaglini and U. DeLiguoro. Sessions and session types: An overview. In *International Workshop on Web Services and Formal Methods*, pages 1–28. Springer, 2009.
10. B. Doerrfeld, C. Wood, A. Anthony, K. Sandova, and A. Laured. The API Economy Disruption and the Business of APIs. Nodic APIs (nordicapis.com), May 2016. Available at `http://nordicapis.com/ebook-release-api-economy-disruption-business-apis`.
11. A. Francalanza, C. A. Mezzina, and E. Tuosto. Reversible choreographies via monitoring in erlang. In *Distributed Applications and Interoperable Systems - 18th IFIP WG 6.1 International Conference, DAIS 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18-21, 2018, Proceedings*, pages 75–92, 2018.
12. D. Gelernter. Generative communication in Linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, 1985.
13. R. Guanciale and E. Tuosto. Realisability of pomsets via communicating automata. In *Proceedings 9th Interaction and Concurrency Experience, ICE 2016, Heraklion, Greece, 8-9 June 2016.*, 2018.
14. R. Guanciale and E. Tuosto. Realisability of pomsets via communicating automata. *Journal of Logic and Algebraic Methods in Programming*, 2019. Accepted for publication; to appear.
15. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. *J. ACM*, 63(1):9:1–9:67, 2016. Extended version of a paper presented at POPL08.
16. H. Hüttel, I. Lanese, V. T. Vasconcelos, L. Caires, M. Carbone, P. Deniélou, D. Mostrous, L. Padovani, A. Ravara, E. Tuosto, H. T. Vieira, and G. Zavattaro. Foundations of session types and behavioural contracts. *ACM Comput. Surv.*, 49(1):3:1–3:36, 2016.
17. H. Hüttel, I. Lanese, V. T. Vasconcelos, L. Caires, M. Carbone, P.-M. Deniélou, D. Mostrous, L. Padovani, A. Ravara, E. Tuosto, et al. Foundations of session types and behavioural contracts. *ACM Computing Surveys (CSUR)*, 49(1):3, 2016.
18. The API-economy. `http://ibm.com/apieconom`.
19. N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon. `http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217`. Working Draft 17 December 2004.
20. J. Lange and E. Tuosto. Synthesising choreographies from local session types. In *CONCUR 2012*, volume 7454 of *LNCS*. Springer, 2012.
21. J. Lange and E. Tuosto. **ChorGram**: tool support for choreographic deveelopment. Available at `https://bitbucket.org/emlio_tuosto/chorgram/wiki/Home`, 2015.
22. J. Lange, E. Tuosto, and N. Yoshida. From Communicating Machines to Graphical Choreographies. In *POPL 15*, pages 221–232, 2015.
23. McAfee. Mcafee security innovation alliance. `https://www.mcafee.com/enterprise/en-us/partners/security-innovation-alliance.html`.
24. McAfee. Threat intelligence exchange recommended workflow. `https://kc.mcafee.com/corporate/index?page=content&id=KB86307`.

25. H. C. Melgratti and E. Tuosto. Summer School on Behavioural Approaches for API-Economy with Applications. https://www.um.edu.mt/projects/behapi/leicester-summer-school-behavioural-approaches-for-api-economy-with-applications, 8-12 July 2019.
26. R. Milner. *Communicating and Mobile Systems: the $\pi$-calculus*. Cambridge University Press, 1999.
27. R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, I and II. *Inf. and Comp.*, 100(1), 1992.
28. R. Neykova, L. Bocchi, and N. Yoshida. Timed runtime monitoring for multiparty conversations. *Formal Aspects of Computing*, 29(5):877–910, 2017.
29. Object Management Group. Business Process Model and Notation. `http://www.bpmn.org`.
30. D. Orenstein. Application Programming Interface. Computer World, Jan. 2000. Available at `http://www.computerworld.com/article/2593623/app-development/application-programming-interface.html`.
31. Z. Qiu, X. Zhao, C. Cai, and H. Yang. Towards the theoretical foundation of choreography. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007*, pages 973–982, 2007.
32. D. Sangiorgi and D. Walker. *The $\pi$-Calculus: a Theory of Mobile Processes*. Cambridge University Press, 2002.
33. E. Tuosto and R. Guanciale. Semantics of global view of choreographies. *J. Log. Algebr. Meth. Program.*, 95:17–40, 2018.