Exploring the Potential of Global Types for Adding a Choreography Perspective to the jABC Framework

Paola Giannini[1]*, Anna-Lena Lamprecht[2] and Tiziana Margaria[2]†
[1]*Computer Science Institute, DiSIT, University of Piemonte Orientale, Alessandria, Italy*
[2]*Lero - The Irish Software Research Centre, University of Limerick, Limerick, Ireland*

Keywords: Choreography, Global Types, Orchestration, jABC, Workflow, Extreme Modeling Framework.

Abstract: We discuss how global types, aka multiparty session types, provide a complementary perspective on workflow models within the jABC modeling framework. On a reference example from the Semantic Web Services Challenge we show how the service orchestrations of jABC workflow applications can be expressed as service choreographies based on global types. Roles, identified with sets of logically related Service-Independent Building Blocks (SIBs), bridge between the two ways of looking at the behavior of systems. We compare the degree of declarativity and robustness in the face of changes of the reference example modeled with the jABC framework with as a global types specification.

1 INTRODUCTION

The eXtreme Model-Driven Design (XMDD) paradigm (Margaria and Steffen, 2009) is a software development methodology that supports automatic or semi-automatic software evolution and management of change by rigorous use of user-level models and refinement throughout the software development process and software life cycle. The current reference implementation of XMDD, the jABC (Steffen et al., 2007), is a framework for service-oriented design and development. With jABC, users develop services and applications by composing reusable building blocks into hierarchical control-flow graph structures that are formally sound yet easy to read and build. From an end-user point of view, all the user interaction happens within an intuitive graphical environment, hardly requiring any classical programming skills for the service *orchestration*.

Choreography is the other principal mechanism for service composition. Choreography-based programming is a powerful paradigm for designing communicating systems where the *flow of communication* is defined abstractly and from a global point of view, instead of separately specifying the behaviour of each participant. A formalization of this paradigm as a type system for a dialect of Robin Milner's π-calculus (Milner et al., 1992) is given in a series of papers by Kohei Honda and others (Honda et al., 1998; Yoshida and Vasconcelos, 2007). The formalization is centered on the notion of *session*, which specifies a sequence of messages exchanged among participants, called *roles*. In (Honda et al., 2008), sessions are described at a global level as the available services, whose types are *multiparty session types*, or *global types*, and at a local level as the protocols seen from the participant perspective whose types are *session types*. These two levels are related: the global processes (thought of as choreographies) and the global types should project to the local ones, producing the actual implementations of the specified system.

Service mediation is a typical application domain for both orchestration and choreography. The *Semantic Web Service Challenge* mediation scenario (SWSC) (Petrie et al., 2009) has been used as a case study for orchestration by many research groups.

In this paper, we refer to this service mediator, introduced in Section 2, and compare its XMDD modeling style with the jABC framework (Section 3) with a global/session types specification (Section 4). We discuss the two approaches in Section 5 and in Section 6 we draw some conclusions, in the context of related work.

*Partly funded by "Progetto MIUR PRIN CINA Prot. 2010LHT4KM" and Torino University/Compagnia San Paolo Project SALT

†Partly supported by Science Foundation Ireland grant 13/RC/2094 and co-funded under ERDF to Lero - the Irish Software Research Centre (www.lero.ie).

Giannini, P., Lamprecht, A-L. and Margaria, T.
Exploring the Potential of Global Types for Adding a Choreography Perspective to the jABC Framework.
DOI: 10.5220/0005799303680376
In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016)*, pages 368-376
ISBN: 978-989-758-168-7

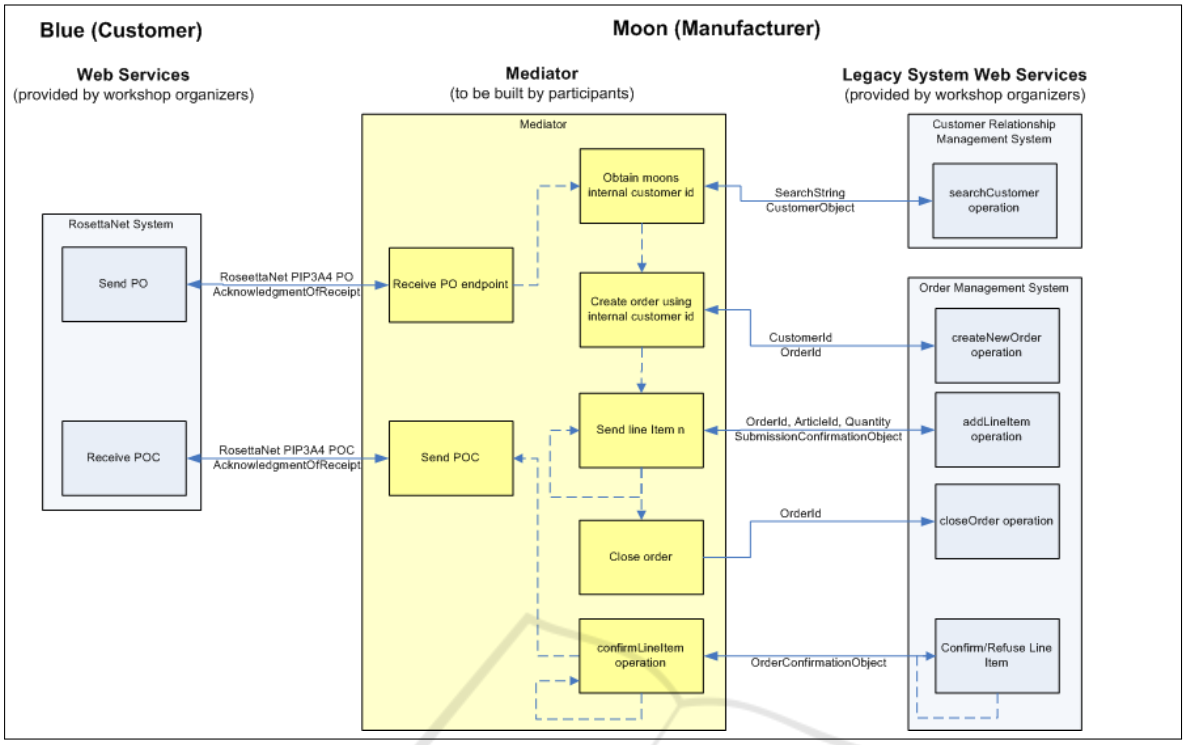

Figure 1: The SWS Mediation Scenario (original picture from the SWS Challenge).

2 THE SWSC MEDIATION SCENARIO

The basic SWSC mediation scenario aims at making a legacy order management system interoperable with external systems that use a simplified version of the RosettaNet PIP3A4 specifications[3]. It requires finding an adequate service composition that adapts two conversation partners where both the interaction protocol and the granularity and format of data mismatch. It consists of the three components in Fig. 1: the **Company Blue** is a customer (service requester) that purchases goods following the RosettaNet specification; the **Legacy System** of the Moon Company instead uses a proprietary system that differs from RosettaNet in data model and message exchange patterns, and the **Mediator**, the sought-for piece of technology providing automatic or semi-automatic mediation between the Blue and the Moon companies. The Mediator must implement the *Purchase Order receiving role* part of the interaction described in the RosettaNet PIP 3A4. In terms of *choreography*, we have a system consisting of at least three roles (Blue, Moon, and Mediator) that need to interoperate via a global protocol. This protocol is abstract, in the sense that it defines the ”shape” of the conversation and messages, but does not prescribe the implementation details (e.g. the concrete steps the software will perform). It describes what the business process community calls a “reference process”, to be later refined towards specific executable solutions, but embodying the global compliance reference. In terms of *orchestration*, we look for a concrete implemented solution for the Mediator component that satisfies the interface requirements with Blue and Moon and manages the internal computations needed to process orders correctly, for any correct shape of the orders. Given the abstraction in terms of shapes, we see here how natural it is in both perspectives to resort to *types* as a semantically precise abstraction: order types, service types, interaction types, conversation types (e.g. Blue/Mediator and Mediator/Moon).

In the SWS Challenge, both the Moon legacy system and the customer Web services (Blue) were provided by the organizers and accessible on the SWSC testbed through public Web services described using WSDL. The services themselves could not be altered by the participants, although their description could be semantically enriched. From a global types point of view, these are two independent roles whose (web)services are, however, physically co-located at one site. In terms of structures amenable to roles, we

[3]http://www.rosettanet.org/PIP3A4

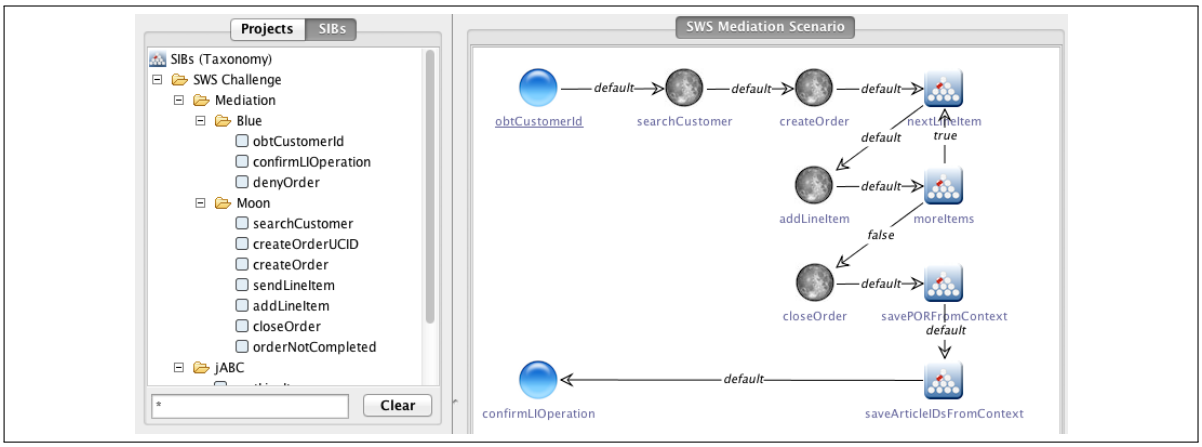

Figure 2: The original mediator scenario in the jABC.

can further observe that the mediator sketch of Fig. 1 requires two services: one from the RosettaNet request to the CloseOrder (called Part 1 in the SWS solutions) and one for the order confirmation (Part 2). They were indeed implemented as two distinct services in our original solutions (Margaria et al., 2008; Margaria et al., 2012). Additionally, to manage its order processing, Moon uses two back-end systems:a Customer Relationship Management system (CRM) and an Order Management System (OMS), that are additional candidates for role refinement.

The Challenge participants were requested to use Semantic Web technologies of their choice to address: the **Data mediation** to map the Blue RosettaNet PIP 3A4 message to the messages of the Moon back-end systems; the **Process mediation** to map message exchanges defined by the RosettaNet PIP 3A4 process to those defined in the WSDL of the Moon back-end systems; the **Conversations** between the systems including data and process mediation on semantic descriptions of messages, thus requiring the transformation from messages used by existing systems to the ontological level.

The following **generic structure** was applied independently by all solutions: (1) Extract the *relevant* information from the *PurchaseOrderRequest* and call Moon's Customer Relation Management (CRM) to find the customer data inside the database (if she already has an account). (2) Use the *CustomerID* to create an order using Moon's Order Management System (OMS). (3) Add *LineItems* as needed and then close the order. (4) Finally the middle layer receives an *OrderConfirmationObject* and sends a *PurchaseOrderConfirmation* back to Blue.

This common reference structure, identified independently of the current work, plays an important role in the global types solution of Sect. 4.

3 ORCHESTRATING THE MEDIATOR IN jABC

In the three years of the SWS Challenge we provided a large set of jABC-based solutions to the mediation scenario problem, with increasing levels of automation and dynamism, discussed in (Margaria et al., 2012). While the first mediators were modeled manually using the jABC's facilities for model-driven design (Margaria et al., 2008), later these models were automatically generated from semantically enhanced abstract specifications with a synthesis algorithm (Margaria et al., 2012).

In XMDD, applications are service compositions (called Service Logic Graphs, or SLGs) that orchestrate along the flow of control basic services in the form of *SIBs - Service-Independent Building Blocks*). Applications are constructed within an intuitive graphical SDK environment, hardly requiring any classical programming skills.

XMDD is largely coding-free: Fig. 2 shows the original solution to the mediator scenario (Part1) in the jABC and the corresponding SIB palette. This SLG orchestrates different roles and respects the generic solution structure previously described. The SIBs are *parameterizable service types*: they are instantiated by dragging them onto the drawing area, where the SLG assembly takes place. This model is compiled to running code via Genesys, jABC's code generation plugin.

The domain-specific model includes the definition of *taxonomies* for the types and the services (SIBs). These taxonomies often group services and types according to particular roles in the application domain: as shown in Fig. 2, Blue's and Moon's SIBs are grouped together and display a suggestive blue/moon icon in the SLG. Therefore, these taxonomies provide

```
module globalProtocol;
global protocol Mediator(role Client, role RosMediator,
        role MoonMediator, role Helper, role CustumerRelation, role Server) {
    do MessageAck<startservice(PurOrderReq)>(Client, RosMediator);
    do MessageReturn<obtCustomerId(PurOrderReq),(SearchString,Tuple)>
        (RosMediator,Helper);
    send(SearchString,Tuple) from RosMediator to MoonMediator;
    do MessageReturn<searchCustomer(SearchString),(CustomerObject)>
        (MoonMediator,CustumerRelation);
    do MessageReturn<createOrderUCID(CustomerObject),(CustomerID)>
        (MoonMediator,Server);
    do MessageReturn<createOrder(CustomerID),(OrderID)>
        (MoonMediator,Server);
    rec CollectItem{
        do MessageReturn<sendLineItem(Tuple),(LineItem)>
            (MoonMediator,Helper);
        choice at MoonMediator {
           do MessageReturn<addLineItem(LineItem),(SubmConfObj)>
            (MoonMediator,Server);
           continue CollectItem;
        } or {
           do MessageReturn<closeOrder(SubmConfObj),(OrderID)>
            (MoonMediator,Server);
       }
    }

    choice at MoonMediator{
        send(PurOrderCon) from MoonMediator to RosMediator;
        do MessageAck<confirmLIOperation(PurOrderCon)>
            (RosMediator,Client);
        } or {
            orderNotCompleted() from MoonMediator to RosMediator;
            do MessageAck<denyOrder()>(RosMediator,Client);
        }
    }
global protocol MessageAck<sig message>(role Sender, role Receiver)
{
    message from Sender to Receiver;
    ack() from Receiver to Sender;
}
global protocol MessageReturn<sig messageIn,sig messageOut>
    (role Sender, role Receiver)
{
    messageIn from Sender to Receiver;
    messageOut from Receiver to Sender;
}
```

Figure 3: The mediator protocol using global types.

through the identified and recognizable roles a useful bridge towards the connection with global types.

In XMDD, these domain models are additionally used by application designers to incrementally express their business knowledge about the models and domain-specific knowledge on the taxonomies. Such information/constraints in CTL are used by the model checker to verify correctness/compliance, and in LTL are used by the synthesis of correct/compliant by construction application SLGs.

4 CHOREOGRAPHING THE MEDIATOR WITH GLOBAL TYPES

Using global types, we provide a choreography that describes the **generic structure** introduced at the end of Section 2. To write the mediator protocol of Fig. 3, we adopt (a subset of) the syntax of Scribble, introduced in (Honda et al., 2011).

We introduce the syntax of Scribble by commenting the protocol in Fig. 3. A protocol is defined by a *name*, a non-empty set of *role parameters*, a possibly empty sequence of *message signature parameters*, and a *global type* specifying how messages are exchanged between the roles involved.

The basic action of a global type is communication, i.e., *sending a message* from one role to another role, $msg(par_1,\ldots,par_n)$ `from` $role_1$ `to` $role_2$. A message has a *name* and may have *parameters*. Message sending is not blocking, and messages sent from one role to another are assumed to arrive in the order they were sent.

The definition of a protocol is preceded by the declaration of the types of the data that are sent with messages, which in our case are the types of the jABC domain platform defined for the mediation scenario. (In Fig. 3 we omitted the definition of these types.) Here, messages correspond mostly to single SIBs, but in more complex examples they could correspond to entire subworkflows that refine the implementation towards an operational executability. Lines $2 \div 35$ define the `Mediator` protocol. The participants involved are declared as *roles* in lines 2 and 3 (the parameters of the protocol). The roles `Client` and `Server` represent the services provided by the challenge organizers, whereas the roles `RosMediator` and `MoonMediator` correspond to `Blue` and `Moon` of the jABC service taxonomy. Role `Customer` provides the communication with the data base of users, and `Helper` is an interface to some general data manipulation services.

In the definition of the `Mediator` protocol we use two communication patterns, specified by the protocols: `MessageAck` (lines $36 \div 40$) and `MessageReturn` (lines $41 \div 46$). Both protocols are generic in the messages specified by the identifiers in the triangular parentheses following the keyword `sig`. `MessageAck` specifies a message send followed by waiting for an acknowledgment from the receiver, and `MessageReturn` specifies a message send followed by waiting for a message back from the sender. This second protocol can be used to return values to the sender, as we will see when it will be instantiated in the body of the `Mediator`.

The `Mediator` protocol starts when the `Client` sends a request with the `PurOrderReq` data and waits for acknowledgement that `RosMediator` has received the message. This is done (line 4) by instantiating the generic protocol `MessageAck` with the message `startService(PurOrderReq)` and associating the formal role parameters with the two roles involved. Then, the role `RosMediator` sends a message to the `Helper` role in order to obtain from the `SearchString` in the `PurOrderReq` the `Tuple` representing the lines of the order that are sent one by one, with the *recursion construct*, from the `MoonMediator` to the `Server` (lines $14 \div 25$). The body of the `rec` construct is executed and if the `continue` statement is found (line 20), the execution of the body is started again. Inside the body of `rec`, after obtaining a single line of order from the `Helper` role, there is a *choice construct* (lines $17 \div 24$) specifying that `MoonMediator` is the role making the choice, and that the interaction continues following either one of the branches, which are separated by `or`. The first branch of the choice (lines $18 \div 20$) says that `MoonMediator` is sending another line of the order to the `Server` and then restarts the body of the `rec`. The other branch (line 22 and 23) specifies that `MoonMediator` closes the order (there are no more lines to be sent). The rest of the protocol description should be obvious.

Global types are projected onto the individual roles producing an implementation of the system. In Fig. 4, we show the projection of the global protocol on `MoonMediator` and `RosMediator`, the most interesting roles. To show the exchanged messages, in the global protocol of Fig. 3, we instantiated all the `MessageAck` and `MessageReturn` subprotocols before doing the projection.

Local protocols see communications from the point of view of a single role. The basic actions are *sending a message*: $msg(par_1,\ldots,par_n)$ `to` *role*, and *receiving a message*: $msg(par_1,\ldots,par_n)$ `from` *role*.

The projection of the global protocol on a given role is obtained by maintaining only the communications that have this role as either the sender or the receiver of the message. Then $msg(par_1,\ldots,par_n)$ `from` $role_1$ `to` $role_2$ produces $msg(par_1,\ldots,par_n)$ `to` $role_2$ in $role_1$ and $msg(par_1,\ldots,par_n)$ `from` $role_1$ in $role_2$. The projection of line 7 of the global protocol of Fig. 3 on `RosMediator` and `MoonMediator` produces line 10 and line 25 of local protocols of Fig. 4.

For a choice construct to be projectable, the first action of each branch must be sending a message from the role making the choice to some other role, moreover, the messages at the beginning of the branches must be distinct. In our global protocol there are two choices: lines $17 \div 24$ and lines $27 \div 34$. Both choices are made by `MoonMediator`. These choices are projected on the local protocol for `MoonMediator` producing the *internal choices* of lines $35 \div 42$ and lines $44 \div 48$. An internal choice means that `MoonMediator` chooses which message to send to whom. The choices are also projected on the roles that are receiving the messages from `MoonMediator`, which are the role `Server` for the choice in lines $17 \div 24$, and the role `RosMediator` for the one in lines $35 \div 42$. In the receiving roles these are *external choices* whose branches start with a receive action from `MoonMediator`. The receiving role must provide a branch for each possible message

received from `MoonMediator`. The projection of the second choice of the global protocol (lines $44 \div 48$ of Fig. 4) on `RosMediator` produces the internal choice of lines $11 \div 19$ of the local protocols of Fig. 3). Note that, the choice of lines $17 \div 34$ of the global protocol does not produce actions in `RosMediator`.

5 DISCUSSION

The native *SLGs* of the jABC and the complementary approach based on *global types* embody two different views of the same specification of the system's behaviour as implemented on the jABC platform for the SWSC service mediator.

The SWSC organizers actually intended to test the robustness of the semantic methods in view of changes and system evolution: a set of successive modifications built upon this initial mediation problem foresaw changes in some aspects of the problem. The evaluation criteria concerned the degree of *declarativity* of the specification (ideally, using semantics the middle layer should be able to autonomously react to changed process specification), and the *robustness* of the solutions: how little needed to be changed manually to satisfy the amended scenarios. Accordingly, the three central dimensions we discuss here are the fitness for change management in light of variability and evolution, the role played to this aim by *roles* and *projections*, and how they foot on the domain modelling underlying the description styles. We briefly discuss them in a bottom-up order.

Domain Modelling. Both specifications require an *initial domain model*: specifying the services needed for the system and organizing data and services in domain-specific taxonomies. The data taxonomy is common to both methods, whereas the service taxonomy differs for the two views. Both descriptions of the behaviour of systems rely on the level of granularity of services offered by the SIBs, and use aggregations of SIBs to compose the required computational abstractions. In jABC, the SIBs are organized in multifaceted taxonomies where facets reflect independent description criteria like functional proximity, provenance (e.g., the providers, here Blue, Moon, or the Mediator - with jABC icon), or SLA-relevant criteria like efficiency or cost. In the case of global types, SIBs are primarily aggregated according to the roles, i.e. according to the participants in the interaction that should enjoy some kind of location proximity. In this sense, we can see a potential correspondence between the provenance facet of the SIB taxonomies and the roles within the global types.

Roles and Projections. The *SLGs* are workflows organized in hierarchical graph structures, and SLGs express with graphical models the *orchestration* of services (SIBs). SLGs primarily focus on the control flow, which is directly depicted in the models and directly executable by means of the Tracer, making the data flow less immediately evident in these models. Both manual SLG design based on the SIB universe, and automatic SLG synthesis from SLTL constraints and taxonomies address problem-specific knowledge and specifications. In a sense, the underlying philosophy is"solving my specific problem" using existing suitable SIBs: a decidedly **customer-oriented** view.

In contrast, *global types* specify a *choreography* where the focus is the collaboration shape. The order of the service execution is only implicitly given by this description, but not fixed. Additionally, it explicitly shows the flow of data. Global type specifications require a suitable identification of sets of logically related services that constitute the *roles* of the interaction. The messages exchanged can be associated with simple SIBs or with sub-workflows composed of SIBs belonging to the same role. From the global type description, the individual behaviour of roles can be generates by projection, as also more generally (correct) traces of executions. Especially this projection capability onto the roles is here of interest: such projections yield localized specifications of suitable service provider behaviours. This is a different and **provider-centric** point of view: given the projections, service providers can check and decide whether they are able to be suitable partners for a specific role in a complex, global choreography, even not knowing what the other partners do or contribute.

This localization capability is interesting for a future enhancement of XMDD's high-level description capabilities, especially with regard to adding change management in long-lived, evolving, collaborative workflows.

Change Management: Variability, Evolution. In both cases, expressing variability and changes is easier than for the typical native web services: since in both our description styles there are no compositions of operations following a predefined protocol of execution, one does not need to define here the kind of low-level adaptation that is needed at the level of the web services.

Variability is expressed in SLGs by means of variation point-SIBs that define the set of possible variants of the workflow. Case by case they are linked to the sub-workflow implementing the chosen variant, in a sort of lightweight discovery of the variant, often also guided by constraints. The taxonomies and the constraint languages are also used to describe such variation points and the choices. In global types, an

```
module localProtocols;
//-----Local RosettaMediator protocol-------
local protocol Mediator_RosMediator at RosMediator(role Client,
    role MoonMediator, role Helper)
{
   startservice(PurOrderReq) from Client;
   ack() to Client;
   obtCustomerId(PurOrderReq) to Helper;
   (SearchString, Tuple) from Helper;
   send(SearchString, Tuple) to MoonMediator;
   choice at MoonMediator{
      send(PurOrderCon) from MoonMediator;
      confirmLIOperation(PurOrderCon) to Client;
      ack() from Client;
   } or {
      orderNotCompleted() from MoonMediator;
      denyOrder() to Client;
      ack() from Client;
   }
}
//-----Local MoonMediator protocol-------
local protocol Mediator_MoonMediator at MoonMediator(role RosMediator,
    role Helper, role CustumerRelation, role Server)
{
   send(SearchString, Tuple) from RosMediator;
   searchCustomer(SearchString) to CustumerRelation;
   (CustomerObject) from CustumerRelation;
   createOrderUCID(CustomerObject) to Server;
   (CustomerID) from Server;
   createOrder(CustomerID) to Server;
   (OrderID) from Server;
   rec CollectItem {
      sendLineItem(Tuple) to Helper;
      (LineItem) from Helper;
      choice at MoonMediator{
         addLineItem(LineItem) to Server;
         (SubmConfObj) from Server;
         continue CollectItem;
      } or {
         closeOrder(SubmConfObj) to Server;
         (OrderID) from Server;
      }
   }
   choice at MoonMediator{
      send(PurOrderCon) to RosMediator;
   } or {
      orderNotCompleted() to RosMediator;
   }
}
```

Figure 4: The local protocols for `RosMediator` and `MoonMediator`.

explicit choice operator specifies on one side that the role at which the choice is made will contain the selection of one possible branch, identified by a message name, and on the other side that whatever expected message arrives, there are roles offering adequate behaviours for all the possible alternatives.

Concerning *evolution*, the scenario changes of the SWS challenge were relatively simple, not adding much to the current insights. A more elaborate application scenario including variability and evolution in complex scientific workflows[4] is currently being investigated. Preliminary results so far seem to confirm that these three dimensions are the key to use the viewpoints' differences in a profitable way. They seem also to be a good basis for being able to use the one or the other viewpoint at need, with a good mapping of the correspondence in both directions.

6 CONCLUSIONS AND RELATED WORK

We showed the use of global types to provide a complementary, choreography-oriented perspective on workflow models within the XMDD paradigm and illustrated it along a benchmark service mediation scenario from the SWSC. The ability to adopt an orchestration-like or a choreography-like point of view at need is important in light of increasingly complex service-oriented workflows, that require a tight

[4]The case study concerns sea level rise consequences in climate change scenarios (Al-Areqi et al., 2014).

and formally well-defined correspondence between the customer-oriented view underlying orchestration and the provider-friendly view of a choreography. We also showed how the underlying domain abstraction of the jABC platform provides a bridge between the different ways of looking at a system behaviour.

The case study illustrates the feasibility in principle of this well-defined correspondence, that we are going to investigate on a larger case study from the climate research domain. While plenty of related work is present on several aspects of formal methods for service-orientation, we see our contribution as demonstrating the practical feasibility of a fielded case study within a general-purpose software engineering development framework.

We consider several extensions to our current setting. The ultimate goal of declarative models and specifications like those we described is to support ease of a system's change management in terms of variation, dynamics, and evolution. Interesting approaches with which we are confronting ourselves address both software product lines and software ecosystems like (Seidl et al., 2014), or concern different kinds of architectural and service provision evolution as in (Inzinger et al., 2013), which align well with our approach (Margaria et al., 2010).

The role of distribution and consistent behaviour of distributed actors in a remote collaboration context is central to cloud computing, Internet of Things, Scientific Computations and Big Data. Context Model Ontologies that support multiple facets and aid in the rule-based expression of consistency and compatibility in dynamic settings have been studied in Lero (Bandara et al., 2015). Even if at the moment we support less than full OWL, such richer ontologies can be used equally well in both approaches and thus are of significant interest to us.

On the description language side, Chor (Montesi, 2013), the choreographic language based on global types designed for the Sensoria platform[5], includes more advanced constructs, such as delegation and the possibility of defining partial choreographies that are interesting in a volatile system. However, in Chor there is no separation between the language in which the roles are written (Jolie) and the choreography specification, which could make a practical separation of layers difficult.

Our ongoing efforts concern the study of specific challenges posed by real life case studies, and the integration of richer description and projection mechanism into the XMDD platform.

[5]http://www.sensoria-ist.eu

REFERENCES

Al-Areqi, S., Kriewald, S., Lamprecht, A.-L., Reusser, D., Wrobel, M., and Margaria, T. (2014). Towards a flexible assessment of climate impacts: The example of agile workflows for the ci:grasp platform. In *ISoLA2014*, LNCS 8803, pp. 420–435. Springer.

Bakera, M., Margaria, T., Renner, C., and Steffen, B. (2009). Tool-supported enhancement of diagnosis in model-driven verification. *Innovations in Systems and Software Engineering*, 5:211–228.

Bandara, K. Y., Wang, M., Pahl, C. (2015). An extended ontology-based context model and manipulation calculus for dynamic web service processes. *Service Oriented Computing and Applications*, 9(2):87–106.

Honda, K., Mukhamedov, A., Brown, G., Chen, T., and Yoshida, N. (2011). Scribbling interactions with a formal foundation. In *ICDCIT 2011*, LNCS 6536, pp. 55–75. Springer.

Honda, K., Vasconcelos, V., and Kubo, M. (1998). Language Primitives and Type Disciplines for Structured Communication-based Programming. In *ESOP'98*, volume 1381 of *LNCS*, pages 22–138. Springer.

Honda, K., Yoshida, N., and Carbone, M. (2008). Multiparty Asynchronous Session Types. In *POPL'08*, pages 273–284. ACM.

Inzinger, C., Hummer, W., Lytra, I., et al. (2013). Decisions, models, and monitoring - A lifecycle model for the evolution of service-based systems. In EDOC 2013, pp. 185–194. IEEE Computer Society.

Margaria, T., Kubczak, C., and Steffen, B. (2012). The xmdd approach to the semantic web services challenge. In *Semantic Web Services*, pp. 233–248. Springer.

Margaria, T. and Steffen, B. (2007). LTL-Guided Planning: Revisiting Automatic Tool Composition in ETI. In *SEW 2007*, pages 214—226. IEEE Computer Society.

Margaria, T. and Steffen, B. (2009). Agile IT: Thinking in User-Centric Models. In ISoLA 2008, Vol.17 of *CCIS*, pp. 490–502. Springer.

Margaria, T., Steffen, B., and Kubczak, C. (2010). Evolution support in heterogeneous service-oriented landscapes. *J. Braz. Comp. Soc.*, 16(1):35–47.

Margaria, T., Bakera, M., Kubczak, C., Naujokat, S., Steffen, B. (2008). Automatic Generation of the SWS-Challenge Mediator with jABC/ABC *Semantic Web Services Challenge. Results from the First Year*, pp. 119-138, Springer, 2008.

Milner, R., Parrow, J., and Walker, D. (1992). A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40, 41–77.

Montesi, F. (2013). *Choreographic Programming*. Ph.D. thesis, IT University of Copenhagen.

Petrie, C., Margaria, T., Lausen, H., and Zaremba, M., editors (2009). *Semantic Web Services Challenge. Results from the First Year*, vol. 8 of *Semantic Web and Beyond*. Springer.

Seidl, C., Schaefer, I., and Aßmann, U. (2014). Integrated management of variability in space and time in software families. In *SPLC '14*, pages 22–31. ACM.

Steffen, B., Margaria, T., Nagel, R., Jörges, S., and Kubczak, C. (2007). Model-Driven Development with the jABC. In *Hardware and Software, Verification and Testing*, LNCS 4383, pp. 92–108. Springer.

Yoshida, N. and Vasconcelos, V. (2007). Language Primitives and Type Disciplines for Structured Communication-based Programming Revisited. In *SecReT'06*, ENTCS 171, pp. 73–93. Elsevier.