Safe Suggestions Based on Type Convertibility to Guide Workflow Composition

Mouhamadou Ba, Sébastien Ferré, Mireille Ducassé

Safe Suggestions Based on Type Convertibility to Guide Workflow Composition

Mouhamadou Ba(✉)[1], Sébastien Ferré[2], and Mireille Ducassé[1]

[1] IRISA/INSA Rennes
20 Avenue des Buttes de Coesmes, 35708 Rennes cedex, FRANCE
[2] IRISA/Université de Rennes 1
263 Avenue Général Leclerc, 35042 Rennes cedex, FRANCE
mouhamadou.ba@irisa.fr

Abstract. This paper proposes an interactive approach that guides users in the step-by-step composition of services by providing safe suggestions based on type convertibility. Users specify the points of the workflow (called the focus) they want to complete, and our approach suggests services and connections whose data types are compatible with the focus. We prove the safeness (every step produces a well-formed workflow) and the completeness (every well-formed workflow can be built) of our approach.

1 Introduction

In a number of domains, particularly in bioinformatics, there is a need for complex data analysis. To that end, elementary data analysis operations, called services, are composed as workflows. Due to the distributed and heterogeneous resources, it is difficult to compose services. Some systems rely on experts who manually develop workflows as scripts. The approach has limitations in usability and scalability [1]. For example, even for experts, the development of scripts quickly becomes heavy and time-consuming. Moreover, developing scripts is, in general, beyond the capacities of domain end-users. Automated approaches are proposed to generate workflows [2]. For example, Kashlev and al. [3] generate executable workflows by matching and transforming input and output data. AI planning techniques are also used to automatically generate compositions of services [4]. Although fully automatic approaches seem appealing, in the context of bioinformatics they are not adapted because users are scientists who generally wish to control the construction of workflows. Furthermore, those approaches require *a priori* complete specifications that are almost impossible to produce in a bioinformatics context. In order to integrate users in the process of composing services, interactive approaches propose support along two axes: verification and suggestions. For example, Kim et al. [5] provide an error scan algorithm that detects errors and provides suggestions to fix them. Their suggestion mechanism relies on global properties (e.g., no cycle, no redundancy) of the workflow but does not focus suggestions to particular points of the workflow the user wants to manage. DiBernado et al. [6] are able to decompose data structures in order to suggest services that can consume parts of those data. However, they do not enable suggestions on multiple points, and do not recompose input data from several parts of the output. The approach of Kim et al. has the same drawback.

In this paper, we propose a guided approach of service composition based on convertibility between input and output types of services. We formally define a workflow model, a suggestion mechanism and possible user actions. The contributions of the paper are manifold. Firstly, users can specify several points where they want to connect a new service. Those points form what we call the focus. The suggestions give only the services that can be safely connected to the focus, and all of them. Hence the number of suggestions that users have to consider for an insertion is significantly reduced. Secondly, all user actions are safe with respect to type convertibility: they lead to a well-formed workflow. There is therefore no need for *a posteriori* type verification. Thirdly, the approach is complete: all workflows that are well-formed with respect to type convertibility can be built. The construction steps can be applied backward or forward (from inputs to outputs, or from outputs to inputs) at any time. Furthermore, the convertibility algorithm, defined in Ba et al. [7], allows to take into account in a fine way the composite structure of data types. A proof-of-concept implementation exists.

Section 2 summarizes our previous work on convertibility between inputs and outputs of services. Section 3 defines in a formal way the principles of our workflow composition approach. Section 4 states properties of safeness and completeness, and Section 5 shows a use case.

2 Type Convertibility

Type convertibility is used to assess compatibility between inputs and outputs of services. It relies on a rule system based on type abstractions More details about convertibility can be found in Ba et al. [7].

Services and Data Type Abstractions. In this paper, we concentrate on the data types of the main parameters of services. The inputs and outputs of services are represented by abstract types defined from an open set of primitive types and a fixed set of type constructors. The type constructors are *Tag*, *Optional*, *Tuple*, *List* and *Union*. *Tag* denotes XML elements, *Optional* denotes not required elements, *Tuple* denotes sequences of elements that have different types, *List* denotes sequences of elements that have the same type, and *Union* denotes union of types. Supported types represent data through annotations, hierarchy and composition, the elementary types being primitive types such as *integer* and *string*.

Rule System. The rule system decides whether types representing service inputs and outputs are convertible. Convertibility is defined as a judgement function $f : S \rightarrow T$, where S is a source type and T a target type. A judgement $f : S \rightarrow T$ holds true if and only if it is possible to build a proof tree with that judgement at the root, and where each node instantiates a rule. This is done by a deduction system that works by structural induction on couples of types (S, T), covering all combinations of type constructors for which convertibility is possible. It uses base judgements on tags and on primitive types, defined according to the application domain. Our system, by applying convertibility on a library of services, detects all possible conversions between input and output types of services. Furthermore it generates executable converters.

3 Our Workflow Composition Approach

Our approach guides a user to construct a workflow step by step, using background knowledge about the services and type convertibility presented in the previous section. To define a workflow, the addition of a service consists of the following 3 steps. First, users indicate the points of the workflow that they want to grow. Second, our system makes suggestions related to those points, using background knowledge and preserving workflow properties. Third, users can select a suggestion to extend the workflow. They can also remove an element of the workflow at any time. In the following, we introduce the components used in our approach, and formalize them.

3.1 Knowledge Base

The knowledge base is composed of types ($\mathcal{T}$), ports (Π), domain services (Σ) and type convertibilities (Λ).

Definition 1. *An abstract port $\pi = (name, type) \in \Pi$ has a name and a type $\in \mathcal{T}$. A service $\sigma = (name, \Pi^{in}, \Pi^{out}) \in \Sigma$ has a $name$, a set of abstract input ports Π^{in}, and a set of abstract output ports Π^{out}. A convertibility $\lambda = (type_1, type_2) \in \Lambda$ has a couple of types such that $type_1$ is convertible to $type_2$.*

In the following, we use dot notation to designate tuple components in definitions, for example $\sigma.\Pi^{out}$ is the set of output ports of service σ.

3.2 Workflow

A workflow is defined by instantiating services and ports, and by linking them.

Definition 2 (workflow). *A workflow is a 3-tuple $W = (P, T, L)$, where*

- P is a set of concrete ports, namely instances of abstract ports. Each concrete port $p = (id, \pi)$ is uniquely identified by id and refers to the port π it instantiates.
- T is a set of tasks, namely instances of services. Each task $t = (id, \sigma, P^{in}, P^{out})$ is an instance of the σ service, it is uniquely identified by id. It defines input ports and output ports ($P^{in}, P^{out} \subset P$) that are, respectively, instances of the input and output ports of σ ($\sigma.\Pi^{in}$, $\sigma.\Pi^{out}$). T contains two special tasks (t_{setter} and t_{getter}) handling the global inputs and outputs of the workflow.
- L is a set of links between ports. Each link $l = (p_{out}, p_{in})$ is defined from an output concrete port to an input concrete port.

We note $W.IP$ and $W.OP$, respectively, the set of input ports and the set of output ports of the tasks of a workflow. The empty workflow is the workflow that has no task and no links, apart the special tasks t_{getter} and t_{setter} and their ports. A workflow W can be abstracted as a dataflow graph.

Definition 3 (dataflow graph). *Given a workflow $W = (P, T, L)$, its dataflow graph $DF(W)$ is defined by the graph $G = (V, E)$, where $V = W.P$ and*
$E = W.L \cup \{(p_{in}, p_{out}) \mid \exists t \in W.T : p_{in} \in t.P^{in} \wedge p_{out} \in t.P^{out}\}$

3.3 Workflow Properties

Properties define constraints on the workflow that have to be satisfied during composition. To define these properties, we use port dependencies from the dataflow graph. In a well-formed workflow (see definition 4), there is no cycle, and an input only consumes data from one output. However, a data from an output may feed in several inputs. For each link, data consumed by the input must be convertible from the output data. We also consider fully-defined workflows where all inputs are connected.

Definition 4. *Considering the dataflow graph, a port p_2 depends on a port p_1 if and only if there is a path from p_1 to p_2 in the dataflow graph. A workflow is* well-formed *if and only if*
(1) no port of the workflow depends upon itself, i.e., there is no cycle in $DF(W)$,
(2) every input port of the workflow is, at most, the target of one link
and (3) every link (p_{out}, p_{in}) verifies convertibility, i.e., $(p_{out}.\pi.type, p_{in}.\pi.type) \in \Lambda$.

3.4 Focus and Suggestions

Suggestions are components proposed to users during composition. Given a workflow $W = (P, T, L)$, we define a focus $F \subseteq P$ as a set of ports of the tasks composing the workflow. The focus defines points of the workflow from which tasks and links can be added. We define $F^{in} = F \cap W.IP$ the input ports of the focus and $F^{out} = F \cap W.OP$ the output ports of the focus. Asking for suggestions with reference to the focus is equivalent to requesting services that can be connected to all focus ports. Suggestions concern on the one hand services whose input and output types are convertible to/from the port types of the focus, and on the other hand links verifying convertibility between ports of the workflow. Suggestions must guarantee that the workflow after insertion is well-formed.

Function $sugg^{\sigma}$ (see definition 5) returns suggested services and the associated links. In the definition of $sugg^{\sigma}$, detected convertibilities between the ports of the focus and ports of a service are described by mappings (φ^{in}) from the set of output ports of the focus to the set of input ports of the service, and (φ^{out}) from the set of input ports of the focus to the set of output ports of the service. The two mappings define links between focus ports and ports of suggested services. For a suggestion to be valid, it is necessary that (1) no input port of the focus depends on a port of the workflow, and (2) no output port of the focus depends on an input port of the focus. The first condition prevents multiple links on input ports, and the second prevents cycles. The function for suggestion of links between existing tasks is not shown but it is similar to the function for suggestion of tasks, it is based on the same properties.

Definition 5 (service suggestions). *Provided that $\forall p_o \in F^{out}, p_i \in F^{in}$, p_o does not depend on p_i, and p_i does not depend on any port of the workflow, the suggested services are defined as follows:* $sugg^{\sigma}(W, F) = \{(\sigma, \varphi^{in}, \varphi^{out}) \mid \sigma \in \Sigma \wedge$
$\varphi^{in} \in F^{out} \rightarrow_{injective} \sigma.\Pi^{in} \wedge \forall p \in F^{out} : (p.\pi.type,\ \varphi^{in}(p).type) \in \Lambda \wedge$
$\varphi^{out} \in F^{in} \rightarrow \sigma.\Pi^{out} \wedge \forall p \in F^{in} : (\varphi^{out}(p).type,\ p.\pi.type) \in \Lambda\}$

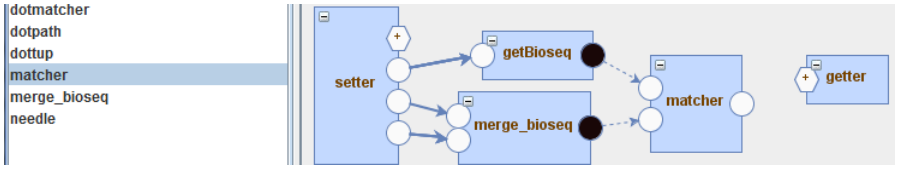

Fig. 1. Insertion of the suggested service 'matcher' in a workflow under construction.

3.5 Transformations

Transformations define authorized actions. The main transformations during workflow construction are insertion and removal of services and links. The insertion of a service corresponds to the addition of a new task to the workflow, and the addition of links between ports of the task and ports of the focus. Insertion of links is similar to insertion of tasks and removal is a simpler case. The services and links to insert in the workflow are chosen from the suggestions. Every insertion of suggested service or link, and every removal produces a well-formed workflow, when starting with a well-formed workflow (see Section 4). We assume that functions $inst^{\pi}$ and $inst^{\sigma}$ define the instantiation of ports and services. Function $insert^{\sigma}$ uses the focus and returns the new workflow after insertion. From a practical point of view, links actually contain converters when needed. Note that users may have to make a choice, when several converters are generated.

Definition 6 (task insertion). *Let* $(\sigma, \varphi^{in}, \varphi^{out}) \in sugg^{\sigma}(W, F)$*, insertion is defined by:* $insert^{\sigma}(W, F, (\sigma, \varphi^{in}, \varphi^{out})) = W'$*, where* $t = inst^{\sigma}(\sigma)$*,*

$$W'.P = W.P \cup t.P^{in} \cup t.P^{out}, \quad W'.T = W.T \cup \{t\},$$
$$W'.L = W.L \cup \{(p, p') \mid p \in F^{out} \wedge p' = inst^{\pi}(\varphi^{in}(p))\}$$
$$\cup \{(p', p) \mid p \in F^{in} \wedge p' = inst^{\pi}(\varphi^{out}(p))\}$$

4 Safeness and Completeness

We state the theorems about the safeness and completeness of suggestions and transformations. Due to lack of space, we only sketch proofs here.

Theorem 1 (safeness & completeness). *Starting from* $W_{\emptyset}$ *(empty workflow) , every sequence of transformations leads to a well-formed workflow. For every well-formed workflow* W*, there is a finite sequence of transformations starting from* $W_{\emptyset}$*.*

Proof (sketch). The empty workflow is well-formed because it does not contain links (e.g., no dependencies). Each transformation on a well-formed workflow produces a well-formed workflow. Thus, by induction, any sequence of transformations leads to a well-formed workflow. We prove the completeness by induction on well-formed workflows with decreasing size, i.e. number of tasks and links. ■

5 Use Case

Figure 1 shows an example of service insertion in a well-defined workflow during composition. Black circles represent the focus. It is composed of two output ports of tasks

$getBioseq$ and $merge\_bioseq$ that have the same type *bioseq*. Considering the focus, the user is interested in services having at least two input ports such that each input is convertible from *bioseq*. According to our system, suggested services at the left of Figure 1 verify that condition. Service *matcher* is one of them and can be inserted. The resulting workflow is well-formed. In practice, ports composing the focus are added one by one. For example, in Figure 1, if we suppose that the user initially adds the output port of *getBioseq*, the initial suggestions contain more suggestions, including services with only one input. When the user adds the output port of *merge_bioseq*, some services are filtered out from the suggestions. The more ports are added to the focus, the more services are filtered out from the suggestions. The use case is created from our proof-of-concept implementation. Our implementation uses about 120 services from the version 5 of Emboss [8]. The GUI enables users to select suggestions and to apply transformations. It is still basic but it is reactive and already complete for the composition of well-formed workflows. A perspective is to take into account others aspects of services and provide ranking using Logical Information Systems [9].

6 Conclusion

This paper provides a guided approach to compose services. The approach is based on convertibility between input and output data types of services. Our system enables users to incrementally specify the focus (a set of ports) of a workflow to be completed; it suggests services and links whose data are compatible with the focus. Our approach produces a well-formed workflow at every step, and enables to produce every well-formed workflow.

References

1. Y. Gil, "Workflow composition: Semantic representations for flexible automation," in *Workflows for e-Science*, pp. 244–257, Springer, 2007.
2. P. Romano, "Automation of in-silico data analysis processes through workflow management systems," *Brief Bioinform*, vol. 9, no. 1, pp. 57–68, 2008.
3. A. Kashlev, S. Lu, and A. Chebotko, "Coercion approach to the shimming problem in scientific workflows," in *IEEE Int. Conf. on Services Computing*, pp. 416–423, 2013.
4. J. Rao and X. Su, "A survey of automated web service composition methods," in *Semantic Web Services and Web Process Composition*, pp. 43–54, Springer, 2005.
5. J. Kim, M. Spraragen, and Y. Gil, "An intelligent assistant for interactive workflow composition," in *Int. Conf. on Intelligent User Interfaces*, pp. 125–131, 2004.
6. M. DiBernardo, R. Pottinger, and M. Wilkinson, "Semi-automatic web service composition for the life sciences using the biomoby semantic web framework," *Journal of Biomedical Informatics*, vol. 41, no. 5, pp. 837–847, 2008.
7. M. Ba, S. Ferré, and M. Ducassé, "Generating data converters to help compose services in bioinformatics workflows," in *Database and Expert Systems Applications*, pp. 284–298, Springer, 2014.
8. P. Rice, I. Longden, and A. Bleasby, "Emboss: the european molecular biology open software suite," *Trends in genetics*, vol. 16, no. 6, pp. 276–277, 2000.
9. S. Ferré, "Camelis: a logical information system to organise and browse a collection of documents," *Int. Journal of General Systems*, vol. 38, no. 4, pp. 379–403, 2009.