Introducing an Artifact-driven language for Service Composition

Willy Kengne Kungne, Georges-Edouard Kouamou, Claude Tangha

Introducing an Artifact-driven language for Service Composition

Willy KENGNE KUNGNE*
Faculty of Sciences, Computer Sciences, University of Yaoundé I
Yaoundé, Cameroon
kengnekungnewilly@yahoo.fr

Georges-Edouard KOUAMOU
National Advanced School of Engineering, University Of Yaoundé I Yaoundé, Cameroon
georges.kouamou@polytechnique.cm

Claude TANGHA
Protestant University of Central Africa,
Yaoundé-Cameroun
Yaoundé, Cameroon
ctangha@gmail.com

ABSTRACT

The most recent service composition approaches rely on the mechanism, which involves scalable and decentralized execution of services. Although some formal tools have been used to this effect, they are influenced by the standard of web service orchestration and choreography based mainly on workflow languages or notation. In this paper, we describe the formal semantics of a novel service composition language through which the services are declaratively composed and executed following a peer-to-peer paradigm. The proposed language named *GSLang* is inspired by the *GAG* (Guarded Attribute Grammars) model that has been defined for the modeling collaborative systems. Pi-calculus is used to define the basic elements of the language and its operational semantics. Then its properties are highlighted through a case study.

KEYWORDS

Dynamic Service Composition, Formal Approach, GAG, Pi-calculus

ACM Reference format:

Willy KENGNE KUNGNE, Georges-Edouard KOUAMOU, and Claude TANGHA. 2019. Introducing an Artifact-driven language for Service Composition. In Proceedings of ArabWIC conference Research (ArabWIC'19). ACM, Rabat, Morocco, 6 pages.
https://doi.org/10.1145/3333165.3333173

1 Introduction

The concept of services in Software Engineering refers to a piece of software, which provides some little functionality to its environment. The most important benefit of services is their interoperability [2]. This feature allows a system to easily leverage the functionalities of another. Service oriented computing has emerged from this vision of software as a promising solution to enhance the functionality of the standard services by composing them into large structures. Complex systems can be built by integrating various independent services. Since most of the approaches are based on business process modeling languages and notations, this study extends also a collaborative case management model so called GAG (Guarded Attribute Grammars) [1] to propose a language for the service composition.

The GAG model defines the workspaces for each user in a formal way through Grammars. It makes it possible to follow the execution of a case in the artifacts and implements strongly coupled mechanisms for the communication of workspaces. This model was proposed as a solution to data-centric workflow modeling [12]. The proposed language (*GSLang*) allows the composite services to be defined on the fly within a workspace, therefore, resulting in a declarative, decentralized, user-centric, data-driven service composition approach.

We define a composite service as a rule of production of a grammar with a left-hand side (LHS) which is the service to define and a right-hand side (RHS) being the services required to realize the LHS service. When the RHS does not exist, then the service is elementary and can be assimilated to a standard protocol of service such as SOAP (Simple Object Access Protocol) or software architecture style as REST (representational state transfer). Each service is guarded by conditions that enable them to be activated. We formalize *GSLang* by defining the concepts generally present in the field of the composition of services such as: variables (parameters), service, service instance, guard, roles, messages and actions. We describe the semantic rules that include the following operations: instantiation, sending and receiving messages, and refinement and choice of services for their execution. This semantic is described using pi-calculus formalism [3]. The choice of this formalism is motivated by the distribution of the peers across a network and their interaction, which is done through dynamic ports whose are created during execution. A service or a peer is seen as a Pi-calculus process. A peer receives and sends the messages. In this logic, a system consists of a set of processes (Peers or Services) that communicate together. During their interactions, asynchronous channels can be created and used to exchange messages. In addition, the channels so-created are included in the messages. The fact that the channels are

*This is the corresponding author

ArabWIC 2019, March 7–9, 2019, Rabat, Morocco

ACM ISBN 978-1-4503-6089-0/19/03…$15.00
https://doi.org/10.1145/3333165.3333173

dynamically created and sent to the peer into the messages, led us to choose the Pi-calculus for our modeling.

The rest of the paper is organized as follows: section 2 presents the concepts of service composition in *GSLang* and a formal semantics for their execution. Section 3 highlights the properties of the *GSLang* through a case study. Section 4 concludes and issues perspectives to this work.

2 FORMAL DESCRIPTION OF A LANGUAGE FOR THE SERVICE COMPOSITION: *GSLANG*

The *GSLang* takes GAG model concepts [1] such as workspace that is assimilated to peer; an activity is assimilated to a composite or simple service. In addition, it promotes distributivity, flexibility and data-driven (Artifact). We want to transport these properties into the world of services.

Generally, the description of a language consists of two parts: the definition of the basics elements and their behaviors. In this section, pi-calculus will be used to this effect. The basic elements for the composition of services are defined as the concepts and the behavior is apprehended through the semantic rules.

In the pi-calculus [3], the processes perform actions, which can be of three forms: the sending of a message over channel x (written $\bar{x}$), the receiving of a message over channel x (written x), and internal actions (written τ), the details of which are unobservable. Send and receive actions are called synchronization actions, since communication occurs when the corresponding processes synchronize. The notion of a transition represents the execution of a process expression. Intuitively, the transition relation tells us how to perform one-step of the process execution. Note that since there can be many ways in which a process executes, the transition is fundamentally nondeterministic. The transition of a process P into a process Q by performing an action α is indicated $P \xrightarrow{\alpha} Q$. The action α is the observation of the transition.

2.1 Basic elements of *GSLang*

The different elements of the *GSLang* are as follows:

2.1.1 ***Terms and variables***. A **variable** is characterized by a letter; it is an entity that may contain a value. **Terms** are variables, values, defined variables (assignments), Boolean expressions or functions on the terms. We define the following element $\bar{\bar{x}}$ by the tuples $(x_1 \dots x_n)$.

$$t ::= x(variable) \mid u(value) \mid x_r(defined\ variable) \mid f(t_0 \cdots t_n)$$

We extend the basic syntax of pi-calculus with Boolean expressions to verify the activation and the validation of a service.

Boolean expressions when specified and evaluated give a Boolean value.

$$e_b ::= true \mid false \mid t_j \leq t_i \mid t_i = t_j \mid e_b \mid e_b \wedge e_b \mid e_b \vee e_b$$

The **assignment** consists in solving for the variables; that is, assigning values to them. A Parameter is an input or an output variable related to a service.

$$x_r ::= \varepsilon \mid x_r\ (x \leftarrow t)(Assignment\ of\ value)$$

2.1.2. ***Service and Service Instance***. A **service** is an entity defined by a unique identifier, input variables (input parameters), output variables (output parameters), guards, post-conditions and a location which represents the associated peer. A service may depend on other services.

$$S ::= id(\bar{\bar{x}}, \overline{e_b}^x)\langle \bar{\bar{y}}, \overline{e_b}^y \rangle[\alpha] \mid id(\bar{\bar{x}}, \overline{e_b}^x)\langle \bar{\bar{y}}, \overline{e_b}^y \rangle[\alpha] \rightarrow s_1 \cdots s_n$$

Such as presented a service is simple or composite. It is characterized by an identifier (its name), input parameters $\bar{\bar{\boldsymbol{x}}}$, output parameters $\bar{\bar{\boldsymbol{y}}}$, possible preconditions on input parameters $\overline{\boldsymbol{e_b}}^{\boldsymbol{x}}$ and possible effects on the output parameters $\overline{\boldsymbol{e_b}}^{\boldsymbol{y}}$. It may be composed of other services $S_1 \dots S_n$. α represents the service location. It should be noted that α may be unnecessary for services on the RHS if they are implemented in the same user space as the services from the LHS.

For reasons of readability, we have preferred the previous notation for services in the paper. In the pi-calculus notation, it corresponds (simple or composite) to:

$$S ::= [\overline{e_b}^x]\alpha(id, \bar{\bar{x}}, p).[\overline{e_b}^y]\ p!\ \bar{\bar{y}} \mid [\overline{e_b}^x]\alpha(id, \bar{\bar{x}}, p).(\nu p_1\ \alpha\langle id_1, \overline{\bar{x}_1}, p_1\rangle \mid \dots \mid S_i \dots \mid \nu p_n\ \alpha\langle id_n, \overline{\bar{x}_n}, p_n\rangle).[\overline{e_b}^y]\ p!\ \bar{\bar{y}}$$

The service S expects $\bar{\bar{\boldsymbol{x}}}$ as the input parameter, when executed, it returns$\bar{\bar{\boldsymbol{y}}}$. It receives the data via the public port of the peer where it is accommodated. When there is a RHS, it calls the services it contains to build *y*. The services on the RHS can be executed in parallel if the data are independent of each other or in sequence if there is dependency hence the parallel operator (|) inside the brackets in bold. When a service call is initiated, a corresponding service instance is created. The same notations can be used for instances.

The **Artifact** or **Service instance** is the concrete representation of a service after the instantiation of the corresponding rule. It allows to track the execution of the service.

$$I ::= id(\bar{\bar{x}}, \overline{e_b}^x)\langle \bar{\bar{y}}, \overline{e_b}^y \rangle[\alpha]$$
$$id(\overline{\bar{x}_r}, \overline{e_b}^x)\langle \bar{\bar{y}}, \overline{e_b}^y \rangle[\alpha] \mid$$
$$id(\overline{\bar{x}_r}, \overline{e_b}^x)\langle \bar{\bar{y}}_r, \overline{e_b}^y \rangle[\alpha] \mid$$
$$id(\overline{\bar{x}_r}, \overline{e_b}^x)\langle \bar{\bar{y}}, \overline{e_b}^y \rangle[\alpha] \rightarrow I_1 \dots I_n \mid$$
$$id(\overline{\bar{x}_r}, \overline{e_b}^x)\langle \bar{\bar{y}}_r, \overline{e_b}^y \rangle[\alpha] \rightarrow I_1 \dots I_n$$

A service instance has several configurations: (i) the input and the output parameters are not yet resolved; (ii) resolved input parameters and output parameters not yet resolved; (iii) resolved input and output parameters. Each element, which appears at the

right-hand side when it exists, should have one of the three previous configurations. The parameters of the service instances are resolved progressively during execution.

2.1.3 ***Message***. **Messages** are variables that transit on the network. They contain global variables (context variables). They are composed of defined variables and/or undefined variables. There are two types of messages: request message (variables in defined inputs and variables in undefined output) and response message (defined input variables and defined output variables).

$$M ::= \overline{\overline{x_r}}\ \bar{y}\ id\ (request) \mid \overline{\overline{x_r}}\ \overline{\bar{y}_r}(response)$$

A **sending message** (request) comprises 3 parts: resolved input $\overline{\overline{x_r}}$, outputs to be resolved $\bar{y}$ and the identifier of the service to which the request is intended. A response message consists of 2 parts: resolved inputs $\overline{\overline{x_r}}$ and resolved outputs $\overline{\bar{y}_r}$. As we will see in the next section, the response is transmitted along a private port created during the request, hence the absence of the service identifier.

2.1.4 ***Action***. An **action** could be any of the following: sending message, receiving message, or silent interpretation of service instances.

$$act ::= vp\ \bar{\alpha}\ \langle M,p\rangle \mid \alpha(M,p) \mid p(M) \mid \bar{p}\langle M\rangle \mid r$$

2.1.5 **Role**. The **peer** or **execution space** (Σ) contains services, instances of services and is characterized by a single public port (its location). Let denotes by S_s the set of services, I_s the set of service instances and α its address (main port or location). Thus, the execution space is characterized by
$\Sigma = (S_s, I_s, \alpha)$.

2.2 Behavioral description

The following operational semantics describe the mechanisms for resolving services, which is broken down into several fundamental operations: instantiation, sending and receiving message, refinement and choice of services.

Instantiation

$$\frac{\Sigma' = \Sigma \cup I, p \notin fn(\Sigma)}{\Sigma: S = id(\bar{\bar{x}})\langle\bar{\bar{y}}\rangle[\alpha] \xrightarrow{\alpha(M,p)} \Sigma': I = id(\overline{\overline{x_r}})\langle\bar{\bar{y}}\rangle[\alpha]}\ C_1$$

$$\frac{\Sigma' = \Sigma \cup I, p \notin fn(\Sigma)}{\Sigma: S = id(\bar{\bar{x}})\langle\bar{\bar{y}}\rangle[\alpha] \to S_1 \dots S_n \xrightarrow{\alpha(M,p)} \Sigma': I = id(\overline{\overline{x_r}})\langle\bar{\bar{y}}\rangle[\alpha] \to I_1 \dots I_n}\ C_2$$

When Σ receives on its main port α the message M and the variable p, it finds the corresponding service S and creates the instance I with the defined input $\overline{\overline{x_r}}$ and the awaited outputs $\bar{\bar{y}}$. I is added to Σ which becomes Σ′. If no service is found, then the operation will not be applied. C_2 is the extended version of C_1, the found service is composed.

Request

$$\frac{\begin{array}{l} I = id(\overline{\overline{x_r}})\langle\bar{\bar{y}}\rangle[\alpha] or \\ = id(\overline{\overline{x_r}})\langle\bar{\bar{y}}\rangle[\alpha] \to I_1 \cdots I_n\ or \\ = I_0 \to I_1 \dots\ id(\overline{\overline{x_r}})\langle\bar{\bar{y}}\rangle[\alpha] \dots I_n \end{array}}{\Sigma: I \xrightarrow{vp\ \bar{\alpha}\ \langle M,p\rangle} \Sigma: I}\ Req$$

The instance I of the space Σ sends the message M on α (main port of another space). This doesn't change the state of the execution space; M is constructed from parameters of the instance to be concretized.

Response

$$\frac{}{\Sigma: I \xrightarrow{p\langle M\rangle} \Sigma: I}\ Resp$$

Response on a private port (p) of a previously sent request.
M=$\overline{\overline{x_r}}\ \bar{\bar{y}}_r$

Refinement

$$\frac{}{\Sigma: I = id(\overline{\overline{x_r}})\langle\bar{\bar{y}}\rangle[\alpha] \xrightarrow{r} \Sigma: I = id(\overline{\overline{x_r}})\langle\overline{\bar{y}_r}\rangle[\alpha]}\ R_1$$

$$\frac{}{\Sigma: I = id(\overline{\overline{x_r}})\langle\bar{\bar{y}}\rangle[\alpha] \xrightarrow{p(M)} \Sigma: I = id(\overline{\overline{x_r}})\langle\overline{\bar{y}_r}\rangle[\alpha]}\ R_2$$

$$\frac{\bar{\bar{x}'} \subseteq \bar{\bar{x}}}{\begin{array}{ll} \Sigma: I = id_0(\overline{\overline{x_r}})\langle\bar{\bar{y}}\rangle[\alpha] \to & \Sigma: I = id_0(\overline{\overline{x_r}})\langle\bar{\bar{y}}\rangle[\alpha] \to \\ I_1 \dots id_i(\bar{\bar{x}'})\langle\bar{\bar{y}'}\rangle[\alpha'] & \xrightarrow{r} I_1 \dots id_i(\overline{\overline{x_r'}})\langle\bar{\bar{y}'}\rangle[\alpha'] \\ \dots I_n & \dots I_n \end{array}}\ R_3$$

$$\frac{\overline{\overline{x''}} \subseteq \bar{\bar{y}'}}{\begin{array}{ll} \Sigma: I = I_0 \to I_1 \dots & \Sigma: I = I_0 \to I_1 \dots \\ id_i(\overline{\overline{x_r'}})\langle\overline{\bar{y}_r'}\rangle[\alpha'] \dots & \xrightarrow{r} id_i(\overline{\overline{x_r'}})\langle\overline{\bar{y}_r'}\rangle[\alpha'] \dots \\ id_j(\overline{\overline{x''}})\langle\overline{\bar{y}''}\rangle[\alpha''] \dots I_n & id_j(\overline{\overline{x_r''}})\langle\overline{\bar{y}''}\rangle[\alpha''] \dots I_n \end{array}}\ R_4$$

$$\frac{}{\begin{array}{ll} \Sigma: I = I_0 \to I_1 \dots & \Sigma: I = I_0 \to I_1 \dots \\ id_i(\overline{\overline{x_r}})\langle\bar{\bar{y}}\rangle[\alpha] \dots I_n & \xrightarrow{p(M)} id_i(\overline{\overline{x_r}})\langle\overline{\bar{y}_r}\rangle[\alpha] \dots I_n \end{array}}\ R_5$$

$$\frac{\bar{\bar{y}} \subseteq \cup\, \bar{\bar{y}_i}}{\begin{array}{ll} \Sigma: I = id(\overline{\overline{x_r}})\langle\bar{\bar{y}}\rangle[\alpha] \to & \Sigma: I = id(\overline{\overline{x_r}})\langle\bar{\bar{y}}\rangle[\alpha] \to \\ id_1(\overline{\overline{x_{1r}}})\langle\overline{\bar{y}_{1r}}\rangle[\alpha] \dots & \xrightarrow{r} id_1(\overline{\overline{x_{1r}}})\langle\overline{\bar{y}_{1r}}\rangle[\alpha] \dots \\ id_n(\overline{\overline{x_{nr}}})\langle\overline{\bar{y}_{nr}}\rangle[\alpha] & id_n(\overline{\overline{x_{nr}}})\langle\overline{\bar{y}_{nr}}\rangle[\alpha] \end{array}}\ R_6$$

The refinement of an instance consists in materializing the parts not yet defined. The action is silent (materialization of the parameters from those already defined in an instance) or the receipt of a response on a private port.
R1: Calling a simple service (automatic or manual)

R2: Receiving information on a private port for the instance of a simple service. This action results in the materialization of the output parameters.
R3, **R4** and **R6:** Allows the definition of the parameters of certain service instances to the right from the parameters already defined. Semantic rules are used at this level to match attributes.
R5: Upon receipt of a response, materialize the part of the service's instance that made the request.

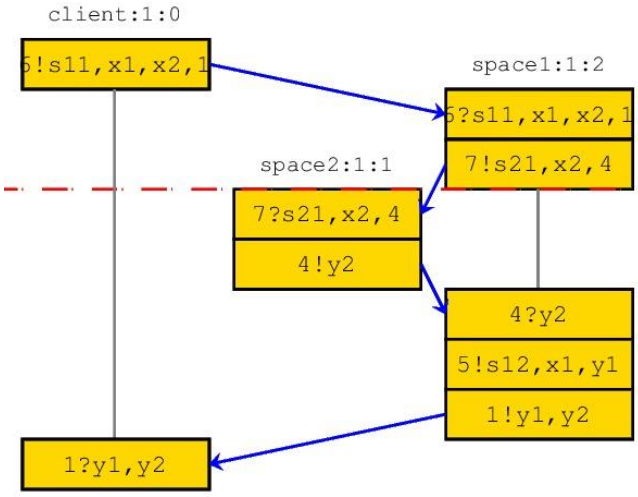

Figure 1: Execution Example

Local choice of Service (LoCh)

$$\frac{id_i(\overline{x_r})\langle\bar{y}\rangle \; match \; to \; I_0{}' \rightarrow I_1{}' \dots I_n{}'}{\begin{array}{c}\Sigma: I = id_0(\overline{x_r})\langle\bar{y}\rangle \rightarrow \\ I_1 \dots id_i(\overline{x_r})\langle\bar{y}\rangle[\alpha] \dots I_n\end{array} \xrightarrow{r} \begin{array}{c}\Sigma: I = id_0(\overline{x_r})\langle\bar{y}\rangle \rightarrow \\ I_1 \dots [I_0{}' \rightarrow I_1{}' \dots I_n{}'] \dots I_n\end{array}} LoCh$$

The selection of a service is made locally when the inputs are defined. Once selected, the previously described operations can be applied.

The emphasis here is on the data that influence the choice of services, their instantiation and their refinements. The execution flow depends on the availability of input and output variable values. In a service execution schema, two tasks are executed in sequence if the entries of one depend on the outputs of the other. They are executed in parallel if the inputs of one do not depend on the outputs of the other. It is for this reason that we have not explicitly defined the parallel operator of the pi-calculus. The conditional choice represents the behaviour of a peer. In a peer, under certain conditions/actions, a service or instance of services can be executed.

3 PROPERTIES OF THE LANGUAGE

This section opens with an example that will serve as a guide in order to highlight the properties of the language.

Following the logic of pi-calculus, a user space is modeled as a process, which holds services materialized by the tasks. In this regard, figure 1 is made up of two processes Σ_1 and Σ_2 representing the user space.

- The space 1 (Σ_1) contains the task s_{11} which starts the process and then decomposes into s_{12} and s_{21} which synchronizes to complete the process. s_{21} is implemented at the level of Σ_2.
- The space 2 (Σ_2) contains the task s_{21}.

Using this language, the process described in Figure 1 presents two user spaces Σ_1 and Σ_2. Initially, Σ_1 contains the services $S_{\Sigma_1} = \{s_{11}, s_{12}\}$, its public port and α_1. Σ_1 services are defined by:

$s_{11}(x_1, x_2\ \{x_1 > 0, x_2 > 0\})\langle y_1, y_2\rangle \rightarrow\ s_{21}\ (x_2)\langle y_2\rangle s_{12}\ (x_1)\langle y_1\rangle$
$s_{12}(x_1)\langle y_1\rangle \rightarrow$

The service s_{11} has a guard $x_1 > 0$ and $x_2 > 0$ and requires s_{21} and s_{12} in order to obtain y_1 and y_2. s_{21} is a remote service implemented in space Σ_2 and s_{12} is a simple local service to Σ_1.

Σ_2 contains the service s_{21} $S_{\Sigma_2} = \{s_{21}\}$ and a public port α_2. The service s_{21} is defined by:

$s_{21}(x_2)\langle y_2\rangle \rightarrow$

Also in the figure 1, a client process c executes the composite service s_{11} of Σ_1. c defines x_1 and x_2, creates a private port p whose value is 1 and sends the message containing s_{11}, x_1, x_2 and p on the public port α_1 (Of value 5) of Σ_1 (the rule C_2 is highlighted). An instance of the s_{11} service will be created in Σ_1 because s_{11} is found and the guard checked. The RHS of s_{11} starts, s_{12} and s_{21} run in parallel since there is no dependency between their parameters. A remote call on Σ_2 will be made to execute s_{21} (the semantic rule Req is used) i.e. creating a private port p_1 of value 4 and sending a message containing s_{21}, x_2 and p_1 on the public α_2 port (Of value 6) of Σ_2.

On arrival at Σ_2, with respect to the input parameters, the s_{21} service is chosen (by applying the C_1 rule), an instance of the service is created and executed. The response (principally y_2) will be sent to Σ_1 via the private port p_1 (applying the *Resp* rule). Σ_1 will refine the previously created instance. The R_6 rule can be used and the p_1 port will be destroyed. Finally, the response will be sent to the client process via the previous private port p (Of value 1).

The asynchronous private ports make it possible to track the execution of the service instances individually. An instance of service may be unavailable for a period of time; when it returns it can continue there where it was suspended. In addition, the services can be redefined at any time even during the execution since they are defined on the fly. For example in Σ_1 we can add s_{13} while s_{11} is running. This is called flexibility by change as defined in [6], contrary to flexibility by definition of existing composition approaches [2][11]. In addition, the services are fully

defined in the form of rules. The rules paradigm has been studied as a declarative approach, presenting the advantages [7] [5] [8] as:

- Adaptability: Given the declarative nature of rule-based service compositions, they can be modified and/or expanded to adapt to context-specific situations. The adaptation of the proposed language in this paper is possible at runtime because each rule (composite service) is identified and loaded when the rule is enacted. RHS not yet enabled can be updated even while running the composite service (LHS). For example in Σ_1 we can add s_{13} while s_{11} is running. s_{11} will become as follows :

$$s_{11}(x_1, x_2\ \{x_1 > 0, x_2 > 0\})\langle y_1, y_2\rangle \rightarrow\ s_{21}\ (x_2)\langle y_2\rangle s_{12}\ (x_1)\langle y_1\rangle\ s_{13}\ (\,)\langle\,\rangle$$

- Flexibility: rule-based compositions are more flexible than BPEL-type compositions, given their ability to pursue other execution paths without having to redefine the composite service and Redeploy it on a service engine. Some languages such as BPEL4WS offer a set of tags (invoke, reply, receive, sequence, choice, flow, etc.) allowing to build the composite service. In our proposition, the definition and the composition of services are described by the declarative rules, while the interaction is implicit through attributes materialized by the transmission of parameters. The private asynchronous ports (dynamic port) created at runtime make the composition more flexible. The execution path of a composite cannot be determined in advance because ports are created and destroyed dynamically as described in the example.
- Formal intuitive semantics: rule-based languages exploit a logical and/or mathematics set of underlying primitives. Formal approaches to reasoning have been proposed [9][10] but all of them use the WS-BPEL process type for their implementation. We propose an intentional definition of services that allows a late concretization of the services, thus favoring a weak coupling with the underlying technology and an adaptation (updating of the rules) of the service even during its execution. Moreover, the proposed language does not refer to any technology. The reasoning can be undertaken on services as we have done in defining operational semantics in section 2 using the pi-calculus.
- Reusability and Distribution: The composite services being defined primarily as rules can be used in different contexts. The services are distributed in different user spaces. The architecture is peer-to-peer. In the example, we have the spaces Σ_1 and Σ_2 located by their public ports α_1 and α_2.

4 RELATED WORK

A service composition language is more flexible when it is based on a declarative paradigm rather than an imperative paradigm as described in [6].

Most of the traditional languages which have been proposed to specify the composition of web services are based on processes, with BPEL as the backbone since all the proposed formalisms are translated into BPEL for their execution [14][2][21]. The disadvantage of this paradigm is that the description of the composite services represented as processes is centralized and difficult to change at runtime.

To overcome this difficulties, some languages have been proposed in order to have more flexibility [15][16][22]. They deal with the semantics of the composition by providing the ability for describing and reasoning over services at runtime [17]. These semantic-based languages are excellent in the discovery, the selection and the automatic composition of services. Their flexibility is limited to searching for missing services or building a composition plan based on a user's query and predefined planning system. It is difficult to add new requirements to the specifications of a composite service when the system is running.

Several declarative approaches to the composition of services have been proposed. The work in [18] defines the rules in the form of if..then clauses, the structure, the data and the constraint rules under the basis of elements such as Activity, Condition, Event, Flow, Provider, Role, and Message. The if..then rules govern how things are to be done in the composition. The if..then rules imply the definition of all the possibilities between the elements of the composition. This is a first step for the flexible composite service definition, but it is defined as an extension of the BPEL notations. To separate the business rules from the BPEL code, Charfi et al. suggested an aspect oriented style (AO4BPEL) [19]. Authors in [20] propose an approach named FARAO. Theyargue that business rules can be used in a service composition without the need for a BPEL framework. This greatly increases the adaptability of the orchestration. At the deployment level, a CA (Condition-Action) rule engine is introduced to support rule-based service composition. To obtain the composite service, an analysis of the services' registry (containing the WSDLs) is performed in order to have dependencies between services and to build CA rules. In CA rules, business rules and constraints will be added. Although using the rules to build the composite services, this approach has an abstraction level of the rules, which are quite low (linked to WSDL). As previous approaches, FARAO focuses on the orchestration on the detriment of distribution and interaction.

The proposed language adopts an independent approach of structured blocks such as BPML promotes by describing a composition service completely with rules, using the GAG formalism.

The adaptation of the proposed model is possible at runtime because each rule in the execution scheme are identified and loaded when the rule is enacted. Since each workspace is considered as an autonomous peer, its proprietary can update the scheme by adding new rules (service declaration) or modify the right hand side of a rule (redefinition of a composite service).

5 CONCLUSIONS AND PERSPECTIVES

This study introduces an artifact-driven language, which can be served as a framework for service composition. In this paper, we have presented a formal description of the basic concepts of this language and their behavior through the semantic rules. An example is shown on two processes to simulate the execution of a composite service. The proposed language benefits from the properties of the data-centric workflow model, it is built upon:

- The composite services are defined declaratively in the form of rules, which provides more flexibility and adaptability.
- The services participating in a composition collaborate in a peer-to-peer style.
- A service elementary or composite can be reused in different application context.

The further works will develop the support software tools for our service composition language such as the services editor, verification and translation tools. In this regard, the selection of a model-checking environment close to pi-calculus is indicated. To meet the challenges raised by the second iteration of service computing, the language shall evolve to cope with the problems of micro-services paradigm [13].

5 ACKNOWLEDGMENTS

This work was realized in the FUSCHIA project with the support of LIRIMA.

REFERENCES

[1] Eric Badouel and al, Active Workspaces: Distributed Collaborative Systems based on Guarded Attribute Grammars, Apply Computing Review, ACM, Vol 15(3), pp 6-34, 2015.
[2] Quan Z. Sheng and al, Web services composition: A decade's overview, Inf. Sci. , 280: 218-238 (2014).
[3] Sangiorgi, D., Walker, D., The pi-calculus: a Theory of Mobile Processes, Cambridge university press, 2003.
[4] Bultan, T. and al, Conversation Specification: A New Approach to Design and Analysis of E-service Composition, ACM 1-58113680-3/03/0005, WWW, 2003.
[5] Weigand, H., van den Heuvel, W.J., Hiel, Rule-based service composition and service-oriented business rule management, International Workshop on ReMoD, (2008).
[6] Mulyar, N., Schonenberg, M., et al., Towards a taxonomy of process flexibility (extended version), , (2007).
[7] Yao, Y., Chen, H., A rule-based web service composition approach, Autonomic and Autonomous Systems (ICAS), 2010 Sixth International Conference, pp. 150-155. IEEE (2010).
[8] Rosenberg, F., Dustdar, S, Business rules integration in bpela service-oriented approach, E-Commerce Technology, 2005. CEC 2005. Seventh IEEE International Conference, pp. 476-479. IEEE (2005).
[9] Zhu, Y., Huang, Z., Zhou, H., Modeling and verification of web services composition based on model transformation, Software: Practice and Experience , 47(5), 709-730 (2017).
[10] Abouzaid, F., Mullins, J., Model-checking web services orchestrations using bp-calculus, Electronic Notes in Theoretical Computer Science , 255, 3-21 (2009).
[11] Sun, Chang-Ai, et al, Automated testing of WS-BPEL service compositions: A scenario-oriented approach, IEEE Transactions on Services Computing, (2015).
[12] COHN, David et HULL, Richard, Business artifacts: A datacentric approach to modeling business operations and processes, IEEE Data Eng. Bull, 32, 3, p. 3-9. (2009).
[13] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin et R., Safina, Microservices: yesterday, today, and tomorrow, In Present and Ulterior Software Engineering. Springer, Cham, pp. 195-216 (2017)
[14] Sabraoui, A., Ettalbi, A., El Koutbi, M., EnNouaary, A., Towards an uml profile for web-service composition based on behavioral descriptions, Journal of Software Engineering and Applications, 5(09), 711 (2012)
[15] Papazoglou, M.P., Heuvel, W.J., Service oriented architectures: approaches, technologies and research issues, The VLDB Journal The International Journal on Very Large Data Bases, 16(3), 389415 (2007)
[16] Yang, H., Zhao, X., Qiu, Z., Pu, G., Wang, S., A formal model forweb service choreography description language (ws-cdl), ICWS06. International Conference on Web Service, pp.893894. IEEE (2006)
[17] Martin, D. et al, Bringing semantics to web services: The owl-s approach, International Workshop on Semantic Web Services and Web Process Composition. Springer, Berlin, Heidelberg., pp. 26-42 (2004)
[18] Orriens, B., Yang, J., Papazoglou, M., A framework for business rule driven service composition, Technologies for E-Services. pp. 1427 (2003)
[19] Charfi, A., Mezini, M., Ao4bpel: An aspect-oriented extension to bpel, World wide web. 10(3), 309344 (2007)
[20] Weigand, H., van den Heuvel, W.J., Hiel, M., Rule-based service composition and service-oriented business rule management, Proceedings of the International Workshop on Regulations Modelling and Deployment (ReMoD08), pp. 112. June (2008)
[21] Lemos, A. L., Daniel, F., Benatallah, B., Web service composition: a survey of techniques and tools, ACM Computing Surveys (CSUR), 48(3), 33. (2016)
[22] Syu, Y., Ma, S. P., Kuo, J. Y., FanJiang, Y. Y. , A survey on automated service composition methods and related techniques, In Services Computing (SCC), 2012 IEEE Ninth International Conference on (pp. 290-297), (2012) (2012, June).