A publish/subscribe approach for implementing GAG's distributed collaborative business processes with high data availability

Maurice Tchoupé Tchendji, Joskel Ngoufo Tagueu

A publish/subscribe approach for implementing GAG's distributed collaborative business processes with high data availability

Maurice TCHOUPÉ TCHENDJI[1] and Joskel NGOUFO TAGUEU[2]

Department of Mathematics and Computer Science
University of Dschang, PO Box 67, Dschang-Cameroon
LIRIMA, FUCHSIA associate team
[1] maurice.tchoupe@univ-dschang.org, ttchoupe@yahoo.fr
[2] jngoufotagueu@yahoo.com

ABSTRACT. With the ever-increasing development of the Internet and the diversification of communication media, there is a growing interest in distributed business process models that focus on exchanged data (or artifact) to control and pilot processes. The Guarded Attribute Grammars (GAG) is one such model; it stands out from the others by the fact that it emphasizes the central place occupied by user decisions during the process execution: it is both data-driven and user-centric. In this paper we present an approach to implementing distributed collaborative business processes modeled using GAG in which communications are done by publish/subscribe with redirection of subscriptions (pub/sub-RS). Pub/sub-RS—which we propose—guarantees high data availability during the process execution, by ensuring that an actor, perceived as a subscriber, will always receive a data he needs to perform a task as soon as it is produced. Moreover, if the data is semi-structured, and is produced collaboratively and incrementally by several actors, its subscribers will be notified as soon as one of its components (a prefix) is produced at the same time they will be subscribed in a transparent way to the remaining components (the suffix).

RÉSUMÉ. Avec le développement toujours croissant d'internet et la diversification des moyens de communication, il est un intèrêt croissant pour les modèles de processus métiers distribués qui mettent l'accent sur les données échangées (ou artefacts) pour controler et piloter les processus. Les grammaires attribuées avec gardes (GAG) est l'un de ces modèles; il se démarque des autres par le fait qu'il met l'emphase sur la place centrale qu'occupe les décisions des utilisateurs lors de l'exécution d'un processus: il est à la fois centré sur les données et sur l'utilisateur. Dans ce papier, nous présentons une approche de mise en œuvre de processus métiers collaboratifs distribués modélisés à l'aide des GAG dans lesquels les communications se font par publish/subscribe avec redirection de souscriptions (pub/sub-RS). Le pub/sub-RS (que nous proposons), garantit une haute disponibilité des données pendant l'exécution des processus en assurant qu'un acteur (vu comme un abonné), recevra toujours une donnée dont il a besoin pour effectuer une tâche dès qu'elle est produite. De plus, si la donnée est semi-stucturée, et produite collaborativement et incrémentalement par plusieurs acteurs, les abonnés seront notifiés dès qu'une de ses composantes (un préfixe) est produite en même temps qu'ils seront abonnés de manière transparente à ses composantes résiduelles (le sufixe).

KEYWORDS : Collaborative business processes, GAG, Artifact, Publish/Subscribe, Subscription redirection, Semi-structured data, Service oriented computing.

MOTS-CLÉS : Processus métiers collaboratifs, GAG, Artifact, Publish/subscribe, Redirection de souscriptions, Données semi-structurées, Calcul orienté service.

1. Introduction

Business processes are processes that represent the activities of companies. Their purpose is to orchestrate activities that contribute to the achievement of organizational goals. The ever-increasing development of the Internet and the diversification of means of communication has led to the emergence of new needs, including the need for distributed process execution. Most business tasks in large organizations are performed collaboratively by actors possibly positioned in remote geographical locations, requiring therefore, the need for rapid information transfer for consistent decision-making.

Generally, collaborative business process management models are either based on the process activity flow [11, 10]; or on the documents exchanged during the process [12, 6, 13]; or on both, as in models centered on the artifact [1] [8, 4, 9, 5]. A disadvantage of these models is that they model the users of the process as second-class actors, when they are not simply ignored. Indeed, although they are very often the main actors of collaboration, they are usually modelled as plain resources performing specific tasks in a context [3]. It is entirely appropriate that collaborative business process models explicitly highlight the roles played by users, as long as their implications would be predominant in collaboration: this is what Badouel et al. propose in the GAG (Guarded Attribute Grammars) model [3, 2], which is a grammatical approach to model collaborative, distributed, data-driven and user-centric business processes.

Intuitively, a GAG is a collection of *semantic rules* [2] or *business rules* describing for a business process how to produce data (synthesized attribute values) from information of the environment (inherited attribute values and user inputs). In the GAG execution model, the artifacts are used to represent and manage the flow of activities, data and the life cycle of processes. They are intentionally modeled by trees whose nodes represent the tasks and have attributes to contain all the information about a process from its creation in the system to its completion. These nodes can be divided into two subsets: the one of the closed nodes corresponding to the completed tasks, and the one of the open nodes corresponding to the pending or running tasks. The choice of how to perform a task associated with a given open node is left to the discretion of the user who performs it, by selecting one of the business rules (from the GAG) applicable to that open node (see section 2). The business rules applicable to an open node at any given time are a function of data previously generated in the process, and data provided by users: it is in this sense that the GAG model is said to be data-driven and user-centric.

The objective of this paper is to present an approach for implementing distributed collaborative business processes modeled using GAG [3] and communicating through a new variant of publish/subscribe [4] called: *publish/subscribe with redirection of subscriptions* abbreviated as *pub/sub-RS*. We are therefore specifically interested in collaborative business processes involving several actors located on different geographical sites, who work in parallel and communicate asynchronously. With the pub/sub-RS protocol, we want to

1. "An artifact is a document that conveys all the information concerning a particular case from its inception in the system until its completion. It contains all the relevant information about the entity together with a lifecycle that models its possible evolutions through the business process" [3]. So we can assimilate it to an active document [1] combining data and processing.
2. A variant of the semantic rules of attribute grammars.
3. In the rest of this manuscript, we will name "GAG processes" business processes modeled using GAG.
4. The publish/subscribe is an asynchronous communication scheme between publishers and subscribers. Subscribers express their interest in data through subscriptions and publishers publish data for subscribers. A subscriber is notified each time a publication corresponds to a subscription he has made.

guarantee a high availability of information (even if it is only partially produced) in order to allow the actors to start processing as soon as possible.

Paper contributions: the major contribution of this paper is the proposal of a scheme of implementation of GAG processes communicating via pub/sub-RS. The pub/sub-RS allows the exchange of potentially semi-structured data (if necessary, its components can even be produced by different peers) asynchronously, incrementally and without intermediaries [5].

Manuscript organization: section 2 provides an overview of business process modeling with GAG. Some formal definitions are also given. The pub/sub-RS protocol as well as our approach to implementing the coupled model GAG - Pub/sub-RS is presented in section 3. Section 4 presents a detailed illustration of the approach, while section 5 is devoted to the conclusion.

2. Business process modeling with GAG

In this section, we present some fundamental concepts of the GAG model for business process modeling. The interested reader can find a more complete presentation of GAG in [3, 2].

2.1. Business process, business rule and artifact

A business process can be interpreted as a task (t_0) to be executed which, depending on its complexity, can be branched/decomposed into subtasks ($t_1, \ldots, t_n$) potentially executed by different actors. Conceptually, this decomposition can be modeled by a rule called *business rule* that can be represented by a production of the form $P : s_0 \to s_1 ... s_n$ expressing the fact that the service s_0 to be invoked to execute the task t_0 must invoke the (sub)services $s_1, \ldots, s_n$ which are the services to be invoked to execute the subtasks $t_1, \ldots, t_n$ of t_0 respectively. We call *s-production* a production having s on the left hand side. For a same service s, we can have several applicable business rules (i.e. several ways to perform a task) and therefore several s-productions. In this case, the choice of the business rule to be applied is up to the actor of the process responsible for executing the task associated with the service (reminder: the execution of a GAG is user-centric).

Each process is associated with an *artifact* modeled by a tree whose nodes are sorted. We write $X :: s$ to indicate that the artifact node X is of sort s; this means that it is an instance of the service s. An artifact is defined by a set of equations of the form $X = P(X_1, ..., X_n)$, indicating that $X :: s$ is a node labeled by the production $P : s \to s_1 \ldots s_n$ and has as successors, the nodes $X_1 :: s_1, \ldots, X_n :: s_n$. A node $X :: s$ that is not defined by any equation is called *open node*; it corresponds to a pending task and will have to be refined (i.e. extended into a subtree, see figure 1) by applying a production corresponding to its sort s (a s-production $P : s \to s_1 \ldots s_n$). This refinement allows to "close" the node X which is now defined by the equation $X = P(X_1, ..., X_n)$ such that $X_1, \ldots, X_n$ are new open nodes created; they have for respective sorts $s_1, \ldots, s_n$ (see figure 1).

To execute a process, we start from an initial artifact reduced to an open node (of the sort of one of the axioms, see definition 2) and refine it by successive application of business rules until we obtain an artifact containing only closed nodes: it is said closed (*closed artifact*).

5. A data produced by a peer X to a peer Y is sent directly to it without passing through a peer Z.

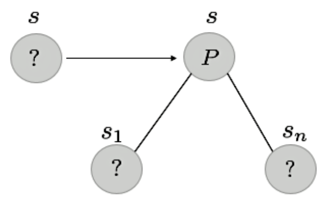

Figure 1. *Refinement of an open node [2]*

2.2. Data flow and configuration of a GAG process

To model the data exchanged between the different services associated with open nodes, more information is attached to the open nodes using attributes. Each sort s is then equipped with a set of inherited attributes and a set of synthesized attributes whose values are terms defined over a ranked alphabet. Recall that a term in this case, is either a variable or an expression of the form $c(t_1,\ldots,t_n)$ where c is a n rank symbol, and $t_1,...,t_n$ are terms. The inherited attributes represent the input values of the services and the synthesized attributes represent the output values. Taking into account the attributes, the complete form of specifying a business rule is as follows:

$$\begin{array}{lcl} s_0(p_1,...,p_n)\langle u_1,...,u_m\rangle & \rightarrow & s_1(t_1^{(1)},...,t_{n_1}^{(1)})\langle y_1^{(1)},...,y_{m_1}^{(1)}\rangle \\ & & \ldots \\ & & s_k(t_1^{(k)},...,t_{n_k}^{(k)})\langle y_1^{(k)},...,y_{m_k}^{(k)}\rangle \end{array} \quad [1]$$

where p_j, u_j, and t_j^l are terms and y_j^l are variables. This new form allows in addition to represent the ramification of a service s_0 into services $s_1,\ldots,s_n$, to also specify the existing dependencies between their data. For example, the rule $s_0(\boldsymbol{p_0}(x,y))\langle \boldsymbol{u_0}(z,t)\rangle \rightarrow s_1()\langle z\rangle s_2(x)\langle t\rangle$ means that the s_0 service must invoke the s_1 and s_2 services by providing s_2 with the x parameter; much more, these services must respectively return the z and t values after their execution. Let's now introduce the notion of *form* which offers a simpler notation for writing business rules.

Definition 1. ***A form*** *of sort s is an expression* $F = s(t_1,...,t_n)\langle u_1,...,u_m\rangle$ *where* t_i *and* u_j *are terms. The terms* $t_1,...,t_n$ *(resp.* $u_1,...,u_m$*) give the values of the inherited (resp. synthesized) attributes attached to the form* F*.*

With this notion, the rule of the equation 1 for example can be rewritten simpler as follows: $F_0 \rightarrow F_1...F_n$ where the F_i are forms. Moreover it also allows us to define more formally a GAG and its configuration at a given time as follows:

Definition 2. ***Guarded Attribute Grammar (GAG)****. Given a set of attribute sorts S, a GAG G is defined by a set* $\mathcal{R}$ *of rules* $R: F_0 \rightarrow F_1...F_k$ *such that* $F_i :: s_i$ $(s_i \in S)$ *are forms, and a set of sorts* $A \subseteq S$ *called axioms:* $G = (\mathcal{R}, \ \mathcal{A})$*. A sort is used (resp. defined) in G if it appears in the right (resp. left) hand side of a rule. Axioms are the sorts that are defined, but not used; they correspond to the services for starting processes. The sort that are used but not defined are called terminals; they correspond to external services. We note N the set of defined sorts and T the set of terminals.*

Definition 3. ***Configuration of a GAG****. A* ***Configuration*** Γ *of a GAG* $G = (\mathcal{R}, \mathcal{A})$ *is a set of sorted nodes X where each node is associated with an equation written in one of the following two forms depending on whether the node is closed or open: for* ***closed nodes,***

$X = R(X_1,...,X_k)$ with $R \in \mathcal{R}$ and $X_1, \ldots, X_n$, the successor nodes of X. For ***open nodes,*** *$X = s(t_1,...,t_n)\langle x_1,...,x_k \rangle$ where X is of sort s; $t_1,...,t_n$ are terms representing the values of the inherited attributes of X, and $x_1,...,x_k$ are variables associated with the synthesized attributes of X.*

The configuration is an extended form of the artifact where data are also taken into account. With the configuration concept, the execution of a GAG process consists of starting from an initial configuration reduced to an open node (of the sort of one of the axioms), and applying business rules to it successively until a configuration with only closed nodes is obtained.

3. Implementation of distributed GAG processes communicating by pub/sub-RS

3.1. The publish/subscribe with subscription redirection: an overview

In a distributed context, each actor in the process has a local GAG that defines how he participates in the process. A local GAG of an actor defines the process services he provides (his axioms), the way he proceeds to perform these services (his business rules) and the services he might need (his terminals). The actor thus participates in the process from his site by applying the business rules of his local GAG. As the process is distributed, an actor may be responsible for executing a service for which certain input data must be provided by other actors (they correspond to some outputs of their services). Similarly, he may also be responsible for the execution of services whose output data are expected by other actors.

To ensure the efficient and without intermediaries exchange of data between actors, we have designed an asynchronous protocol for data exchange between actor based on the publish/subscribe called *publish/subscribe with redirection of subscriptions* in short *pub/sub-RS*. The pub/sub-RS is a new variant of the publish/subscribe protocol particularly suitable for the exchange of semi-structured data whose components can be produced incrementally by different actors. As in the classic publish/subscribe, any actor has the possibility to subscribe to one or more events in order to be notified as soon as a publisher generates an occurrence of an event corresponding to one of his subscriptions [7]. However, unlike the traditional publish/subscribe, in the pub/sub-RS, since the exchanged data are potentially semi-structured and can be produced incrementally (by collaboration of several actors), they are also delivered incrementally. In fact, as soon as a prefix x of a data to which an actor has subscribed is produced, it is immediately sent to him simultaneously as he (the actor) is subscribed (subscription redirection) in a transparent way to the *residue* (the suffix) of the initial data.

The pub/sub-RS's operating mode is therefore as follows: each time an actor needs a data to be produced by another actor, he must subscribe to it; if it is produced incrementally, the prefix produced must each time be sent to him at the same time as he is subscribed to the residue. More concretely, if an actor A needs a data d_b to be produced by an actor B, then a subscription of A to the data d_b must be stored on the actor B's site in a subscription list provided for this purpose and, A will be notified as soon as B produces d_b. If d_b is semi-structured and is produced incrementally, then A will be notified as soon as d_b components are produced. For example, if d_b is a list of the form $d_b = d_1b : d_sb$, then A will be notified as soon as d_1b is generated by B and automatically subscribed to the residual data $d_s b$ (which will not necessary be produced by B). It is the fact that an actor A subscribes to a data d and is then subscribed to the residual data of d ($d_s b$ in the current example) that we call *redirection of subscriptions*.

3.2. Workspace of an actor

An actor's workspace is represented by two data structures: the local configuration Γ of his local GAG and the list of subscriptions on the data he must produce called LS (figure 2 shows an illustration of a workspace). The local configuration of an actor Γ informs him about the tasks in which he participates and the data necessary to perform them. Thus all the data he handles are associated with local variables of his configuration. However, in case of sharing data, a same data can be associated with many variables belonging to different configurations. We say in that case that those variables have the same publication identifier. We define the *publication identifier* of a variable x—noted $\dot{x}$ —as the global unique name of the data intended to be stored there. For example, in the previous description, the data d_b can be stored in the configuration of A in the variable x_a and in the configuration of B in the variable x_b; the variables x_a and x_b in that case must have the same publication identifier ($\dot{x_a} = \dot{x_b}$) since they correspond to local names of d_b.

The subscription list LS is the set of subscriptions on the data that the actor must produce and publish. A subscription is a pair of the form $(\dot{x}, b)$ where $\dot{x}$ is the global unique name (i.e a publication identifier) of the data to which the subscription relates and b is the identifier of the remote actor who subscribes.

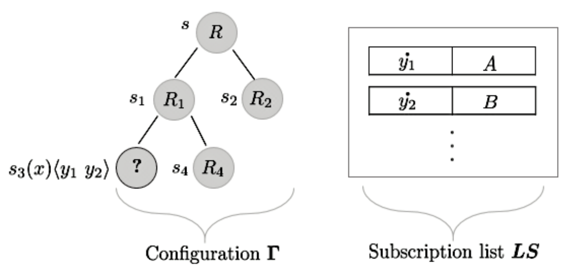

Figure 2. *Illustration of an actor's workspace with two subscriptions on output values*

3.3. Local application of a business rule

Each actor in the process works locally using the business rules contained in his local GAG. In the following, we will assimilate an actor to the site on which he operates by designating him by the identifier of his site. Let's consider a site named a with a configuration Γ and a list of subscriptions LS. Let's also consider $X = F$ an open node of Γ and $R = F_0 \rightarrow F_1 \ldots F_k$ a business rule such that $F_i' s$ variables are disjoint from those of Γ. Apply a business rule R to X is equivalent to sequentially performing the following four operations (see fig. 3): 1) update the local configuration Γ; 2) notify the sites subscribing to the data that have been produced; 3) update the subscription list LS; 4) Call remote services (if necessary). Let's now precisely present the treatments carried out by each of these operations:

(1) Update from Γ to Γ'. The application of R to node X refines it into open child nodes $X_1,...,X_n$ associated with sub-services $s_1,...,s_n$ of $F_1,...,F_n$. Thus, it is necessary to add to the configuration the new nodes created that can be refined by the current actor (the other nodes will be transformed into remote service calls) and close the node X. The new configuration is then Γ':

$$\begin{array}{rcl}\Gamma' & = & \{X = R(X_1,...,X_k)\} \\ & \cup & \{X_i = F_i\sigma \mid X_i :: s_i,\ s_i \in N\} \\ & \cup & \{X' = F \mid (X' = F) \in \Gamma,\ \ X' \neq X\}\end{array}$$

where $\sigma =$**match**[6](F_0, X) is a substitution that matches the input values of F_0 to the input values of F and the output values of F to the output values of F_0. For each variable x of the forms $F_i,\ 0 \leq i \leq n$, we assign its name to the data that will be stored there: $\dot{x} = x$. To ensure that this name is globally unique, one can simply use a local name generator that creates variable names prefixed by the site identifier.

Remark. *We consider that a business rule R is only applicable to an open node $X = s(t_1,...,t_n)\langle y_1,...,y_m\rangle$ if the $t_i, 1 \leq i \leq n$ are completely defined values, i.e. no longer contain variables.*

(2) Subscriber notifications. The application of the rule allows new data to be generated using substitution σ. An equation of the form $(x = t)$ in σ means that the variable x is now assigned the value t. By doing so, all sites that have subscribed to the data represented by x must be notified. These are the sites b as it exists a subscription $(\dot{x}, b)$ in LS. All notifications are therefore: $NS = \{(x = t,\ b) \mid (x = t) \in \sigma$ and $(\dot{x}, b) \in LS\}$, where each element $(x = t,\ b)$ is a notification meaning that the site b (when $b \neq a$) will receive the equation $(\dot{x} = global(t))$; x is the (local) variable associated with the data to which b had subscribed and t is its value; the function $global(t)$ renames the variables in t by their publication identifiers: $global(t) = t[y/\dot{y}]$. When for an element $(x = t,\ b)$ of NS we have $b = a$, no message is sent, it means that the current site a has produced a data to which it is itself subscribed. In that case, we simply update the local configuration with the value of x: $\Gamma' = \Gamma[y/t \mid \dot{y} = \dot{x}\,]$.

(3) Update of the subscription list LS to LS'. Once the notifications have been made, the local subscription list must be updated: subscriptions on data that have already been sent to subscribers (OLS) are removed, and new subscriptions (NLS) from sites related to dependencies between the data defined in the rule are added. The new local subscription list on the site a is therefore: $LS' = LS \setminus OLS\ \cup\ NLS$ where

$$\begin{array}{lcl}OLS & = & \{\ (\dot{x}, b) \mid (x = t, b) \in NS\ \}\ \textit{and} \\ NLS & = & \bigcup\limits_{X_i :: s_i, s_i \in N} NLS(X_i),\ N \text{ is the set of defined sorts;}\end{array}$$

$NLS(X_i)$ is the set of subscriptions on the data to be produced by a new non-terminal node X_i. It is defined by: $NLS(X_i) = \bigcup\limits_{x \in out(X_i)} \{\ (\dot{x}, b) \mid b \in SUBS_{R,\sigma,NS}(x)\ \}$ where $out(X)$ is the set of variables associated with the synthesized attributes of X and $SUBS_{R,\sigma,NS}(x)$ is the set of sites that need the value of variable x; the following paragraph describes how

6. The function *match* is defined in [3]. It returns a substitution σ itself consisting in two main substitutions σ_{in} and σ_{out}. For a specific refinement of a node $X = F$ into child nodes $X_1 = F_1$ to $X_n = F_n$ via a rule $F_0 \rightarrow F_1...F_n$, σ_{in} matches the input values of F_0 to the input values of F and σ_{out} matches the output values of F to the output values of F_0. $\sigma = \sigma_{out} \cup \sigma_{in}\sigma_{out}$ [3].

this set is constructed.

Computation of subscriber sites to a variable. The set of sites to be subscribed to a variable x of a new node is computed from three parameters: the business rule R whose application allowed to create the variable x, the substitution σ produced following the application of the rule and all notifications NS generated by the application of R. It is noted $SUBS_{R,\sigma,NS}(x)$ and is created from the combination of two sets:

$$\begin{array}{rcl} SUBS_{R,\sigma,NS}(x) & = & Brother_{R,\sigma}(x) \\ & \cup & Redirect_{NS}(x) \end{array}$$

with :

– $Brother_{R,\sigma}(x)$: all the sites that will execute a service with x as input variable. If $R = F_0 \to F_1 \ldots F_k$ then all the new nodes created by the rule application are $Node_{R,\sigma} = \{\ X_i = F_i\sigma \mid 1 \le i \le k\ \}$ and we have $Brother_{R,\sigma}(x) = \{\ b \mid (X = F) \in Node_{R,\sigma},\ x \in in(X),\ provider(X) = b\ \}$ where $in(X)$ is the set of inherited attributes (input variables) of X; for a node $X :: s$, $provider(X)$ returns the identifier of the site that provides the service s (it is equal to the current site when s is a defined sort).

– $Redirect_{NS}(x)$: the set of subscriptions created by the redirection of subscriptions. Since the redirection of subscriptions occurs when a data d to which sites had subscribed is partially produced, sites that had subscribed to d (they are in NS[7]) must therefore be subscribed to its residual data. If the term t is the partial produced value of d, then the residual data of d correspond to the variables of t: $Redirect_{NS}(x) = \{\ b \mid (y = t,\ b) \in NS,\ x$ is a variable of $t\ \}$.

(4) Remote service calls. For each new open node created $X :: s$ such that s is a terminal, a service call must be made to the site providing the service s (s is an axiom of its local GAG). This service call must also contain the subscription list for the data to be produced by the service. A service call therefore consists in sending a message $m = (X = F, LST_X)$, where X is the node representing the service, F the form associated with X and LST_X the list of subscriptions to be transferred to the site that will refine the node X. Service calls to be made are extracted from the set $I = \{(X_i = F_i\sigma, LST_{R,\sigma,NS}(X_i), b) \mid X_i :: s_i,\ s_i \in T,\ b = provider(X_i)\}$ where $LST_{R,\sigma,NS}(X) = \bigcup_{x \in out(X)} \{(\dot{x}, b) \mid b \in SUBS_{R,\sigma,NS}(x)\}$.

Each element $(X = F, LST, b) \in I$ means that the current site a will send the service call $(X = global(F), LST)$ to the site b.

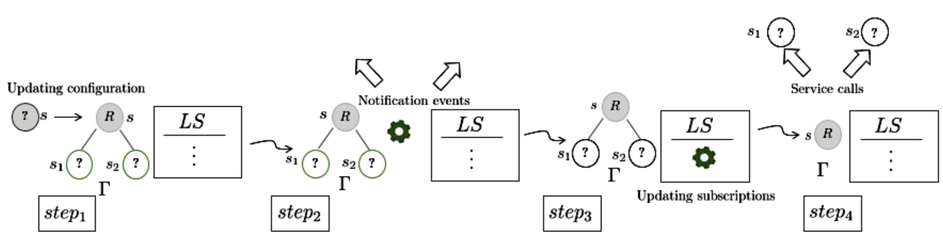

Figure 3. *Different execution steps when applying rule*

7. Recall that NS is the set of notifications created by the application of the rule.

3.4. Processing of messages

Messages received by a site correspond to service call messages or data production notifications.

When a site receives a service call, (a) it updates its local configuration by creating a new local node matching the remote node received via the service call. The variables created in the new local node have the same publication identifiers as those of their correspondents in the remote node received (this will be used for notifications). After that, (b) subscriptions from remote sites are added to the local subscription list. The following formulas (fa) and (fb) summarize these treatments in equational form:
if $m = (Y = s(t_1,...,t_n)\langle y_1,...,y_q\rangle,\ LS_Y)$ is the service call message received, then :

$(fa) : \Gamma' = \Gamma\ \cup\ \{\ \bar{Y} = s(\bar{t_1},...,\bar{t_n})\langle \bar{y_1},...,\bar{y_q}\rangle\ \}$ with $\bar{t} = t[x/\bar{x}]$, $\bar{Y}$ a new local node matching the node Y; the $\bar{x}$ and $\bar{y}_j$ are new local variables created such that $\dot{\bar{x}} = x$, $\dot{\bar{y}}_j = y_j$;

$(fb) : LS' = LS \cup LS_Y$.

In the case of receiving a notification, the site merely updates its local configuration using the publication identifier to know which variable of the configuration to update. The following formula summarizes this treatment in equational form:
if $m = (x = t)$ is the notification message received, then:

– $\Gamma' = \Gamma[y/\bar{t} \mid \dot{y} = x]$ where $\bar{t} = t[z/\bar{z}]$, the $\bar{z}$ being new variables such that $\dot{\bar{z}} = z$;

– $LS' = LS$.

4. Illustration

In this section we illustrate the use of the pub/sub-RS communication protocol in a GAG process through an example. The figure 4 shows the different sites involved and how the subscription computation is used to update their subscription lists when a rule is applied.

Example. *Let's consider a process with five actors A, B, C, D and E such that the actor A has the configuration* $\Gamma_A = \{\ X = s_A()\langle d_A\rangle\ \ \}$ *and the local GAG* G_A *containing a business rule* $R : s_A()\langle \boldsymbol{sum}(d_B, 5, d_C)\rangle \to s_B()\langle d_B\rangle\ s_C(d_B)\langle d_C\rangle$ *(*s_B *provided by B and* s_C *provided by C). Suppose in addition that the subscription list of A is* $LS_A = \{\ (\dot{d}_A, D), (\dot{d}_A, E)\ \}$*. If A applies the business rule R on the node X of* Γ_A *then we will have* $\sigma_{in} = \emptyset$ *and* $\sigma = \sigma_{out} = \{\ d_A = \boldsymbol{sum}(d_B, 5, d_C)\ \}$*;* d_B *and* d_C *being new variables with* $\dot{d}_B = d_B$ *and* $\dot{d}_C = d_C$*. Since* d_A *was partially produced, the actors D and E must be notified of the production of this partial data. To do this, we use the set* $NS = \ \{\ (d_A = \boldsymbol{sum}(d_B, 5, d_C), D),\ (d_A = \boldsymbol{sum}(d_B, 5, d_C), E)\ \}$ *containing the notifications generated by the application of the rule. In addition, D and E must be subscribed to the residual data* d_B *and* d_C *of* d_A *:* $Redirect_{NS}(d_B)$*=* $Redirect_{NS}(d_C)$ *=* $\{\ D,\ E\ \}$*. Finally, as the actor C provides the service* s_C *which has* d_B *in its input, he must be subscribed to* d_B *:* $Brother_{R,\sigma}(d_B) = \{\ C\ \}$*.*

5. Conclusion

In this paper, we have presented an approach of implementing GAG's distributed collaborative business processes in which communication is handled via a new variant

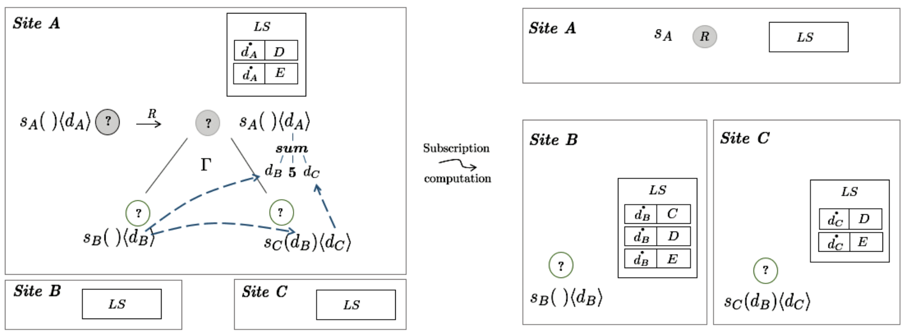

Figure 4. *Illustration of the application of a business rule in a distributed context with the pub/sub-RS*

of publish/subscribe protocol called publish/subscribe with redirection of subscriptions (pub/sub-RS). The main advantage of this protocol is that it permits, in real time, to inform data subscribers on the evolution of the processing operations by incrementally transferring to them any data they have subscribed to. This can encourage the early initiation of other operations if they are lazy (*lazy evaluation*): the degree of parallelism and the speed in decision-making are thus improved. A direct perspective of this work could be to provide an environment that facilitates the edition and deployment of GAG business processes communicating with pub/sub-RS. The idea would be to design a DSL (Domain Specific Language) to easily write a GAG specification, and the tools for the generation and deployment of the software components necessary for the distributed execution.

References

[1] Serge Abiteboul, Omar Benjelloun, and Tova Milo. Positive active xml. In *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 35–45, New York, NY, USA, 2004. ACM.

[2] Eric Badouel, Loïc Hélouët, Georges-Edouard Kouamou, and Christophe Morvan. A grammatical approach to data-centric case management in a distributed collaborative environment. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, pages 1834–1839, New York, NY, USA, 2015. ACM.

[3] Eric Badouel, Loïc Hélouët, Georges-Edouard Kouamou, Christophe Morvan, and Nsaibirni Robert Fondze, Jr. Active workspaces: Distributed collaborative systems based on guarded attribute grammars. *SIGAPP Appl. Comput. Rev.*, 15(3):6–34, October 2015.

[4] Elio Damaggio, Alin Deutsch, and Victor Vianu. Artifact systems with data dependencies and arithmetic. *ACM Trans. Database Syst.*, 37(3):22:1–22:36, 2012.

[5] Elio Damaggio, Richard Hull, and Roman Vaculín. On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles. *Inf. Syst.*, 38(4):561–584, 2013.

[6] J. Dang, C. Toklu, K. Hampel, and U. Enke. Human workflows via document-driven process choreography. In *2008 International MCETECH Conference on e-Technologies (mcetech 2008)*, pages 25–33, Jan 2008.

[7] Patrick Th. Eugster, Pascal Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[8] Richard Hull. Artifact-centric business process models: Brief survey of research results and challenges. In *On the Move to Meaningful Internet Systems: OTM 2008, OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, Monterrey, Mexico, November 9-14, 2008, Proceedings, Part II*, pages 1152–1163, 2008.

[9] Anil Nigam and Nathan S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.

[10] OASIS. Web services business process execution language version 2.0. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf`, April 2007.

[11] OMG. Bpmn specification, business process model and notation. `http://www.bpmn.org/`.

[12] Nelly Schuster, Christian Zirpins, Stefan Tai, Steve Battle, and Nils Heuer. A service-oriented approach to document-centric situational collaboration processes. In *18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, WETICE 2009, Groningen, The Netherlands, 29 June - 1 July 2009, Proceedings*, pages 221–226, 2009.

[13] Lin Zhao, Jianping Xing, and Lingguo Meng. The research and realization of a new workflow model with step-task two layers based on document. In *Proceedings of The 1st IEEE Asia-Pacific Services Computing Conference, APSCC 2006, December 12-15, 2006, Guangzhou, China*, pages 285–292, 2006.