# A Grammatical Approach to Data-centric Case Management in a Distributed Collaborative Environment

Eric Badouel, Loïc Hélouët, Georges-Edouard Kouamou, Christophe Morvan

# A Grammatical Approach to Data-centric Case Management in a Distributed Collaborative Environment

Eric Badouel
Inria and LIRIMA
Campus de Beaulieu
35042 Rennes, France
eric.badouel@inria.fr

Loïc Hélouët
Inria
Campus de Beaulieu
35042 Rennes, France
loic.helouet@inria.fr

Georges-Edouard Kouamou
ENSP and LIRIMA
BP 8390, Yaoundé, Cameroon
georges.kouamou@lirima.org

Christophe Morvan
Université Paris-Est
UPEMLV, F-77454
Marne-la-Vallée, France
christophe.morvan@u-pem.fr

## ABSTRACT

This paper presents a purely declarative approach to artifact-centric case management systems. Each case is presented as a tree-like structure; nodes bear information that combines data and computations. Each node belongs to a given stakeholder, and semantic rules govern the evolution of the tree structure, as well as how data values derive from information stemming from the context of the node. Stakeholders communicate through asynchronous message passing without shared memory, enabling convenient distribution.

## Keywords

Business Artifacts, Case Management, Attribute Grammars

## 1. INTRODUCTION

Case-management consists in assembling relevant information during short collaborative processes that may involve human stakeholders. It is frequently addressed using the notion of *Business Artifacts*, also known as *business entities with lifecycles*, as proposed in [8, 6, 3]. 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.

This paper presents a declarative model for the specification of artifact-centric case management systems where the stakeholders interact according to an asynchronous message-based communication schema. Case-management usually consists in assembling relevant information by calling *tasks*, which may in turn call subtasks, etc. Case elicitation needs not be implemented as a sequence of successive calls to subtasks, and several subtasks can be performed in parallel. To allow as much concurrency as possible in the execution of tasks, we favor a *declarative* approach where task dependencies are specified without imposing a particular execution order.

Attribute grammars are particularly adapted to that purpose. The model proposed in this paper is called *Guarded Attributed Grammars* (GAG). A GAG is an extension of an attribute grammar [7, 9] using a notation reminiscent of unification grammars. It is made of production rules, that are used to complete documents (via a standard rewriting process) and at the same time synthesize and propagate information via the attributes of the grammar.

A production of a grammar is as usual described by a left hand side, indicating a non-terminal to expand, and a right hand side, describing how to expand this non-terminal. We furthermore interpret a production of the grammar as a way to decompose a task (the symbol in the left-hand side of the production) into sub-tasks associated with the symbols in its right-hand side. The semantics rules basically serve as a glue between the task and its sub-tasks by making the necessary connections between the corresponding inputs and outputs (associated respectively with inherited and synthesized attributes).

In this declarative model, the lifecycle of artifacts is left implicit. Artifacts under evaluation can be seen as incomplete structured documents, i.e., trees with *open nodes* corresponding to parts of the document that remain to be completed. Each open node is attached a so-called *form* interpreted as a service call. A form consists of a task together with some inherited attributes (data resulting from previous executions) and some synthesized attributes. The latter are variables subscribing to the values that should emerge from task execution.

Productions are *guarded* by patterns occurring at the inherited positions of the left-hand side symbol. Thus a production is enabled at an open node if the patterns match with the corresponding attribute values appearing in the form. The evolution of the artifact thus depends both on previously computed data (stating which production is enabled) and the stakeholder's decisions (choosing a particular production amongst those which are enabled at a given moment, and inputting associated data). Thus GAGs are both *data-driven* and *user-centric*.


*SAC'15* April 13-17, 2015, Salamanca, Spain.

ACM 978-1-4503-3196-8/15/04...$15.00.
http://dx.doi.org/10.1145/2695664.2695698

Data manipulated in guarded attributed grammars are of two kinds. First, the tasks communicate using *forms* which are temporary information used for communication purpose only. Second, *artifacts* are structured documents that record the history of cases (log of the system). An artifact grows monotonically (we never erase information) and every part of it is edited by a unique stakeholder (the owner of the corresponding nodes), hence avoiding edition conflicts. These properties are instrumental to obtain a simple and robust model that can easily be implemented on a distributed asynchronous architecture.

This paper is organized as follows : Section 2 introduces the formal notations for GAGs. Section 3 gives a semantics to the model in terms of rewriting steps. Section 4 briefly states some formal properties of the model, before conclusion. Proofs of theorems are not provided in this paper, but can be found in an extended version, available at `http://hal.inria.fr/hal-00990007`.

## 2. GUARDED ATTRIBUTE GRAMMARS

This section introduces *Guarded Attributed Grammars*, a grammatical notation for case management. It is inspired by the work of [4] relating attribute grammars with logic programming. Throughout the paper, the term *case* will designate a concrete instance of a given business process. We will use the editorial process of an academic journal as a running example to illustrate the various notions and notations. A case for this example is the editorial processing of a particular article submitted to the journal.

The case is handled by various actors involved in the process, the so-called *stakeholders*, namely the editor in chief, an associate editor and some referees. We associate each case with a document, called an *artifact*, that collects all the information related to the case from its inception in the process until its completion. When the case is closed this document constitutes a full history of all the decisions that led to its completion.

We interpret a case as a problem to be solved, that can be completed by refining it into sub-tasks using business rules. This notion of business rule can be modeled by a *production* $P : s_0 \leftarrow s_1 \cdots s_n$ expressing that task $s_0$ can be reduced to subtasks $s_1$ to $s_n$. If several productions with the same left-hand side $s_0$ exist then the choice of a particular production corresponds to a decision made by some designated stakeholder. For instance, there are two possible immediate outcomes for a submitted article: either it is validated by the editor in chief and it enters the evaluation process of the journal or it is invalidated because its topic or format is not adequate. This initial decision can be reflected by the two following productions:

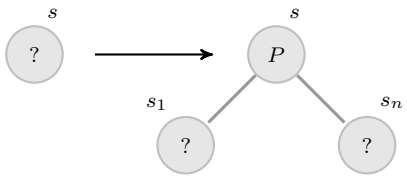
validate : **Proposed_submission** $\leftarrow$ **Submission**
invalidate : **Proposed_submission** $\leftarrow$

If $P$ is the unique production having $s_0$ in its left-hand side, then there is no real decision to make and such a rule is interpreted as a logical decomposition of the task $s_0$ into substasks $s_1$ to $s_n$. Such a production will be automatically triggered without human intervention.

Accordingly, we model an artifact as a tree whose nodes are sorted. We write $X :: s$ to indicate that node $X$ is of sort $s$, where a sort denotes a category of objects sharing common features. An artifact is given by a set of equations of the form $X = P(X_1, \ldots, X_n)$, stating that $X :: s$ is a node labeled by production $P : s \leftarrow s_1 \cdots s_n$ and with successor nodes $X_1 :: s_1$ to $X_n :: s_n$. In that case node $X$ is said to be a *closed* node defined by equation $X = P(X_1, \ldots, X_n)$ (we henceforth assume that we do not have two equations with the same left-hand side). A node $X :: s$ defined by no equation (i.e. that appears only in the right hand side of an equation) is an *open node*. It corresponds to a pending task.

The lifecycle of an artifact is implicitly given by a set of productions:

1. The artifact initially associated with a case is reduced to a single open node.
2. An open node $X$ of sort $s$ can be *refined* by choosing a production $P : s \leftarrow s_1 \ldots s_n$ that fits its sort. The open node $X$ becomes a closed node $X = P(X_1, \ldots, X_n)$ under the decision of applying production $P$ to it. In doing so the task $s$ associated with $X$ is replaced by $n$ subtasks $s_1$ to $s_n$ and new open nodes $X_1 :: s_1$ to $X_n :: s_n$ are created accordingly.




3. The case has reached completion when its associated artifact is closed, i.e. it no longer contains open nodes.

However, plain context-free grammars do not model the interactions and data exchanged between the various tasks associated with open nodes. To overcome this problem, we attach additional information to open nodes using *attributes*. Each sort $s \in S$ comes equipped with a set of *inherited* attributes and a set of *synthesized* attributes. Values of attributes are given by *terms* over a ranked alphabet. Recall that such a term is either a variable or an expression of the form $c(t_1, \ldots, t_n)$ where $c$ is a symbol of rank $n$, and $t_1, \ldots, t_n$ are terms. In particular a constant $c$, i.e. a symbol of rank 0, will be identified with the term $c()$. We will denote by $var(t)$ the set of variables used in term $t$.

DEFINITION 2.1. *A* **form** *of sort $s$, denoted by $F :: s$, is an expression*

$$F = s(t_1, \ldots, t_n)\langle u_1, \ldots, u_m \rangle$$

*where $t_1, \ldots, t_n$ (respectively $u_1, \ldots, u_m$) are terms over a ranked alphabet —the alphabet of attribute's values— and a set of variables $var(F)$. Terms $t_1, \ldots, t_n$ give the values of the* **inherited attributes** *and $u_1, \ldots, u_m$ the values of the* **synthesized attributes** *attached to form $F$.*

We can now define productions where the left-hand and right-hand sides of a rule are defined using forms. More precisely, a production is of the form

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

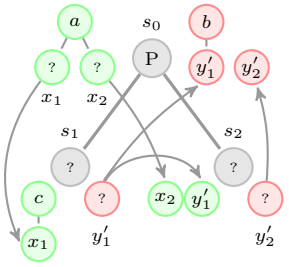
where the $p_i$'s, the $u_j$'s, and the $t_j^{(\ell)}$'s are terms and the $y_j^{(\ell)}$'s are variables. The forms in the right-hand side of a production are *service calls*, namely they are forms $F =$

$s(t_1, \ldots, t_n)\langle y_1, \ldots, y_m\rangle$ where the synthesized positions are (distinct) variables $y_1, \ldots, y_m$ (i.e., they are not instantiated). The rationale is that we invoke a service by filling in the inherited positions of the form (the entries) and by indicating the variables that expect to receive the results returned by the service (the subscriptions).

Any open node is now attached a service call. The corresponding service is supposed to *(i)* construct the tree that will refine the open node and *(ii)* compute the values of the synthesized attributes (i.e., it should return the subscribed values). A service is enacted by applying productions. More precisely, a production such as the one given in formula (1) can apply in an open node $X$ when its left-hand side matches with the service call $s_0(d_1, \ldots, d_n)\langle y_1, \ldots, y_m\rangle$ attached to node $X$. For that purpose the terms $p_i$'s are used as patterns that should match the corresponding data $d_i$'s. When the production applies, new open nodes are created and they are respectively associated with the forms (service calls) in the right-hand side of the production. The values of $u_j$'s are then returned to the corresponding variables $y_j$'s that had subscribed to these values. For instance applying production

$$P : s_0(a(x_1, x_2))\langle b(y_1'), y_2'\rangle \leftarrow s_1(c(x_1))\langle y_1'\rangle \;\; s_2(x_2, y_1')\langle y_2'\rangle$$

to a node associated with service call $s_0(a(t_1, t_2))\langle y_1, y_2\rangle$ gives rise to the substitution $x_1 = t_1$ and $x_2 = t_2$. The two newly-created open nodes are respectively associated with the service calls $s_1(c(t_1))\langle y_1'\rangle$ and $s_2(t_2, y_1')\langle y_2'\rangle$ and the values $b(y_1')$ and $y_2'$ are substituted to the variables $y_1$ and $y_2$ respectively.




A Guarded Attributed Grammar (GAG for short) is defined as a set of production rules of the form $P : F_0 \leftarrow F_1 \ldots F_k$, where all $F_i's$ are forms.

Definition 2.2 (Guarded Attribute Grammars). *A **guarded attribute grammar** is a set of productions. In each production, $P : F_0 \leftarrow F_1 \cdots F_k$, the $F_i :: s_i$ are forms, furthermore, the inherited attributes of left-hand side $F_0$ are called the **patterns** of the production. The values of synthesized attributes of forms $F_1 \cdots F_k$, in the right-hand side, are variables. These occurrence of variables together with the variables occurring in the patterns are called the* **input occurrences** *of variables. We assume that each variable has* **at most one input occurrence**.

The inputs are associated with (distinct) variables and the value of each output is given by a term using these variables. We refer to these correspondences as the **semantic rules**.

For our running example, a GAG defining the editorial process can be defined as follows: A stakeholder has a specific **role** in the editorial process: he can be an author, the editor in chief, an associate editor or a referee. Each role is associated with a set of services and a set of productions explaining how each service is provided. For instance an associate editor provides the service **Submission**$(article)\langle decision\rangle$ consisting in returning an editorial decision about an article submitted to the journal. The corresponding productions are listed in Table 1. The first two productions mean that an associate editor makes an editorial decision about a submitted paper on the basis of the evaluation reports produced by two different referees. He can ask a report from a reviewer through an invocation of the external service **ToReview**$(article)\langle answer\rangle$. The productions that govern the actions of a reviewer are given in Table 2.

Productions Decline$(msg)$ and Accepts$(msg)$ reflect a nondeterministic choice of a reviewer. They are also a way to input new data by assigning a particular message Msg to variable *msg* resulting in the respective attribute values No(Msg) or Yes(Msg, *report*).

One can group the productions of Table 1 and Table 2 using an additional parameter *reviewer* to make as many disjoint copies of the specification given in Table 2 as there are individuals playing the role of a referee. The resulting set of productions (where call to external services have been eliminated) is given in Table 3. Similarly one has as many instances of the productions in Table 1 as there are associate editors in the editorial board. In the complete specification one should therefore add an additional parameter *associateEditor* to distinguish between all associate editors. If the specification is large and contains many different roles the resulting global grammar can be quite complex. Yet, it is still possible to build an equivalent monolithic grammar without external service calls.

## 3. BEHAVIOR OF GAGS

Attribute grammars are traditionally applied to abstract syntax trees which can be produced by some parsing algorithm during a previous stage. The semantic rules are then used to decorate the nodes of the tree by attribute values. In our setting the generation of the tree and evaluation of attributes, using the semantic rules, are intertwined since the input tree represents an artifact under construction.

We consider collaborative systems relying on a distributed memory consisting of the current artifacts. A configuration of this memory can be represented as follows:

Definition 3.1 (Configuration). *A* **configuration** $\Gamma$ *is an $S$-sorted set of nodes $X \in nodes(\Gamma)$ each of which is associated with a defining equation in one of the following form where $var(\Gamma)$ is a set of variables associated with $\Gamma$:*

**Closed node:** $X = P(X_1, \ldots, X_k)$ *where $P : F_0 \leftarrow F_1 \ldots F_k$ is a production of the grammar and $X :: s$, and $X_i :: s_i$ for $1 \leq i \leq k$. Production $P$ is the* **label** *of node $X$ and nodes $X_1$ to $X_n$ are its* **successor nodes**.

**Open node:** $X = s(t_1, \ldots, t_n)\langle x_1, \ldots, x_m\rangle$ *where $X$ is of sort $s$ and $t_1, \ldots, t_k$ are terms with variables in $var(\Gamma)$ that represent the values of the inherited attributes of $X$, and $x_1, \ldots, x_m$ are variables in $var(\Gamma)$ associated with its synthesized attributes.*

*Each variable in $var(\Gamma)$ occurs at most once in a synthesized position.*

We identify a substitution $\sigma$ on a set of variables $x_1, \ldots, x_k$, called the *domain* of $\sigma$, with a system of equations $x_i = \sigma(x_i)$. The set $\mathrm{var}(\sigma) = \bigcup_{1 \leq i \leq k} \mathrm{var}(\sigma(x_i))$ of variables of $\sigma$ is disjoint from the domain of $\sigma$. Conversely a system of equations $\{x_i = t_i\}_{1 \leq i \leq k}$ defines a substitution $\sigma$ with

**Table 1: Acting as an associate Editor**

$$\begin{array}{l}
\text{DecideSubmission} : \mathbf{Submission}(article)\langle decision\rangle \leftarrow \quad \mathbf{Evaluate}(article)\langle report_1\rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{Evaluate}(article)\langle report_2\rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{Decide}(report_1, report_2)\langle decision\rangle \\
\text{MakeDecision}(decision) : \mathbf{Decide}(report_1, report_2)\langle decision\rangle \leftarrow \\
\text{AskReview}(reviewer) : \mathbf{Evaluate}(article)\langle report\rangle \leftarrow \quad \mathbf{WaitReport}(answer, article)\langle report\rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \texttt{Call}(reviewer, \mathbf{ToReview}(article)\langle answer\rangle) \\
\text{CaseNo}\langle msg\rangle : \mathbf{WaitReport}(\text{No}(msg), article)\langle report\rangle \leftarrow \mathbf{Evaluate}(article)\langle report\rangle \\
\text{CaseYes}\langle msg\rangle : \mathbf{WaitReport}(\text{Yes}(msg, report), article)\langle report\rangle \leftarrow
\end{array}$$

**Table 2: Acting as a reviewer**

$$\begin{array}{rcl}
\text{Decline}(msg) & : & \mathbf{ToReview}(article)\langle \text{No}(msg)\rangle \leftarrow \\
\text{Accept}(msg) & : & \mathbf{ToReview}(article)\langle \text{Yes}(msg, report)\rangle \leftarrow \mathbf{Review}(article)\langle report\rangle \\
\text{MakeReview}(report) & : & \mathbf{Review}(article)\langle report\rangle \leftarrow
\end{array}$$

$\sigma(x_i) = t_i$ if it is in *solved form*, i.e., none of the variables $x_i$ appears in some of the terms $t_j$. In order to transform a system of equations $E = \{x_i = t_i\}_{1\leq i\leq k}$ into an equivalent system $\{x_i = t'_j\}_{1\leq j\leq m}$ in solved form one can iteratively replace an occurrence of a variable $x_i$ in one of the right-hand side term $t_j$ by its definition $t_i$ until no variable $x_i$ occurs in some $t_j$. This process terminates when the relation $x_i \succ x_j \Leftrightarrow x_j \in \text{var}(\sigma(x_i))$ is acyclic. Then the resulting system of equations $SF(E) = \{x_i = t'_i\}_{1\leq i\leq n}$ in solved form does not depend on the order in which the variables $x_i$ have been eliminated from the right-hand sides. When the above condition is met we say that the set of equations is *acyclic* and that it *defines* the substitution associated with its solved form.

The composition of two substitutions $\sigma, \sigma'$ is denoted by $\sigma\sigma'$ and defined by $\sigma\sigma' = \{x = t\sigma' \mid x = t \in \sigma\}$. Similarly, we let $\Gamma\sigma$ denote the configuration obtained from $\Gamma$ by replacing the defining equation $X = F$ of each open node $X$ by $X = F\sigma$.

We now define more precisely when a production is enabled at a given open node of a configuration and the effect of applying the production. First note that variables of a production are formal parameters whose scope is limited to that production. They can injectively be renamed in order to avoid clashes with variables names appearing in a configuration. Therefore we shall always assume that the set of variables of a production $P$ is disjoint from the set of variables of a configuration $\Gamma$ when applying production $P$ to $\Gamma$. As informally stated in the previous section, a production $P$ applies in an open node $X$ when its left-hand side $s(p_1, \ldots, p_n)\langle u_1, \ldots u_m\rangle$ matches with the definition $X = s(d_1, \ldots, d_n)\langle y_1, \ldots, y_m\rangle$, i.e., the service call attached to $X$ in $\Gamma$.

First, the patterns $p_i$ should match with the data $d_i$ according to the usual pattern matching operation given by the following inductive statements

$$\begin{array}{l}
\mathbf{match}(c(p'_1, \ldots, p'_k), c'(d'_1, \ldots, d'_{k'})) \text{ with } c \neq c' \text{ fails} \\
\mathbf{match}(c(p'_1, \ldots, p'_k), c(d'_1, \ldots, d'_k)) \;=\; \sum_{i=1}^{k} \mathbf{match}(p'_i, d'_i) \\
\mathbf{match}(x, d) = \{x = d\}
\end{array}$$

where the sum-substitution, $\sigma = \sum_{i=1}^{k} \sigma_i$, of substitutions $\sigma_i$ is defined and equal to $\bigcup_{i\in 1..k} \sigma_i$ when all substitutions $\sigma_i$ are defined and associated with disjoint sets of variables. Note that since no variable occurs twice in the whole set of patterns $p_i$, the various substitutions $\mathbf{match}(p_i, d_i)$, when defined, range over disjoint sets of variables. Note also that $\mathbf{match}(c(), c()) = \emptyset$.

Definition 3.2. *A form $F = s(p_1, \ldots, p_n)\langle u_1, \ldots u_m\rangle$* **matches** *with a service call $F' = s(d_1, \ldots, d_n)\langle y_1, \ldots, y_m\rangle$ (of the same sort) when*

1. *the patterns $p_i$'s match with the data $d_i$'s, defining a substitution $\sigma_{in} = \sum_{1\leq i\leq n} \mathbf{match}(p_i, d_i)$,*
2. *the set of equations $\{y_j = u_j\sigma_{in} \mid 1 \leq j \leq m\}$ is acyclic and defines a substitution $\sigma_{out}$.*

*The resulting substitution $\sigma = \mathbf{match}(F, F')$ is given by $\sigma = \sigma_{out} \cup \sigma_{in}\sigma_{out}$.*

Definition 3.3 (Applying a Production). *Let $P = F \leftarrow F_1 \ldots F_k$ be a production, $\Gamma$ be a configuration, and $X$ be an open node with definition $X = s(d_1, \ldots, d_n)\langle y_1, \ldots, y_m\rangle$ in $\Gamma$. We assume that $P$ and $\Gamma$ are defined over disjoint sets of variables. We say that $P$ is* ***enabled*** *in $X$ and write $\Gamma[P/X\rangle$, if the left-hand side of $P$ matches with the definition of $X$. Then applying production $P$ in $X$ transforms configuration $\Gamma$ into $\Gamma'$, denoted as $\Gamma[P/X\rangle\Gamma'$, where:*

$$\begin{array}{rcl}
\Gamma' & = & \{X = P(X_1, \ldots, X_k)\} \\
& \cup & \{X_1 = F_1\sigma, \ldots, X_k = F_k\sigma\} \\
& \cup & \{X' = F\sigma \mid (X' = F) \in \Gamma \;\wedge\; X' \neq X\}
\end{array}$$

*where $\sigma = \mathbf{match}(F, X)$, and $X_1, \ldots, X_k$ are new nodes added to $\Gamma'$.*

Thus the first effect of applying production $P$ to an open node $X$ is that $X$ becomes a closed node with label $P$ and successor nodes $X_1$ to $X_k$. The latter are new nodes added to $\Gamma'$. They are associated respectively with the instances of the $k$ forms in the right-hand side of $P$ obtained by applying substitution $\sigma$ to these forms. The definitions of the other nodes of $\Gamma$ are updated using substitution $\sigma$ (or equivalently $\sigma_{out}$). This update has no effect on the closed nodes because their defining equations in $\Gamma$ contain no variable.

One can show that applying a production $P$ in an open node $X$ of a configuration $\Gamma$ with $\Gamma[P/X\rangle\Gamma'$ cannot create a variable with several occurrences in synthesized position, i.e. the resulting set of equations $\Gamma'$ is also a configuration. Thus applying an enabled production defines a binary relation on configurations.

**Table 3: Making a decision on a submitted paper**

DecideSubmission : **Submission**$(article)\langle decision\rangle \leftarrow$ **Evaluate**$(article)\langle report_1\rangle$
**Evaluate**$(article)\langle report_2\rangle$
**Decide**$(report_1, report_2)\langle decision\rangle$
MakeDecision$(decision)$ : **Decide**$(report_1, report_2)\langle decision\rangle \leftarrow$
AskReview$(reviewer)$ : **Evaluate**$(article)\langle report\rangle \leftarrow$ **WaitReport**$(answer, article)\langle report\rangle$
**ToReview**$(reviewer, article)\langle answer\rangle)$
Decline$(msg)\langle reviewer\rangle$ : **ToReview**$(reviewer, article)\langle$No$(msg)\rangle \leftarrow$
Accept$(msg)\langle reviewer\rangle$ : **ToReview**$(reviewer, article)\langle$Yes$(msg, report)\rangle \leftarrow$ **Review**$(reviewer, article)\langle report\rangle$
MakeReview$(report)\langle reviewer\rangle$ : **Review**$(reviewer, article)\langle report\rangle \leftarrow$
CaseNo$\langle msg\rangle$ : **WaitReport**$($No$(msg), article)\langle report\rangle \leftarrow$ **Evaluate**$(article)\langle report\rangle$
CaseYes$\langle msg\rangle$ : **WaitReport**$($Yes$(msg, report), article)\langle report\rangle \leftarrow$

DEFINITION 3.4. *A configuration $\Gamma'$ is **directly reachable** from $\Gamma$, denoted by $\Gamma[\rangle\Gamma'$, whenever $\Gamma[P/X\rangle\Gamma'$ for some production $P$ enabled in node $X$ of configuration $\Gamma$. Furthermore, a configuration $\Gamma'$ is **reachable** from configuration $\Gamma$ when $\Gamma[*\rangle\Gamma'$ where $[*\rangle$ is the reflexive and transitive closure of relation $[\rangle$.*

As already mentioned, an artifact is refined by applying a production to one of its open node. However we also need means to initiate cases. To this extent, we define interfaces for GAGs, that describe how services can initialize new artifacts.

DEFINITION 3.5. *The **interface** of a GAG is given by a subset $\mathcal{I}$ of forms $F = s(t_1, \ldots, t_n)\langle x_1, \ldots, x_m\rangle$, called the **service calls**, where the synthesized positions are (distinct) variables $x_1, \ldots, x_m$. This set is closed by substitutions whose domains are disjoints from the set of synthesized variables, namely $F\sigma \in \mathcal{I}$ whenever $F \in \mathcal{I}$ and $\sigma$ is a substitution with $\sigma(x_j) = x_j$ for $1 \leq j \leq m$. The invocation of the service produces a new artifact reduced to a single open node defined by $F$, it is associated with **initial configuration** $\Gamma_0 = \{X_0 = s(t_1, \ldots, t_n)\langle x_1, \ldots, x_m\rangle\}$. A **reachable configuration** of a guarded attribute grammar is a configuration reachable from one of its initial configurations.*

# 4. FORMAL PROPERTIES OF GAGS

Allowing rewriting using attributes that take values over unbounded terms makes the model very expressive. Unsurprisingly, this expressive power implies that some formal properties are undecidable.

A specification is **sound** if every case can reach completion no matter how its execution started. A case is a service call in the interface of the GAG (Definition 3.5) which already contains all the information coming from the environment of the guarded attribute grammar.

DEFINITION 4.1. *Given a guarded attribute grammar with its interface, a **case** $c = s(t_1, \ldots, t_n)\langle x_1, \ldots, x_m\rangle$ is an element of the interface such that $\mathrm{var}(t_i) \subseteq \{x_1, \ldots, x_m\}$. Stated otherwise a case is, but for the variables with a synthesized value, a closed instance of a service.*

DEFINITION 4.2. *A configuration is **closed** if it contains only closed nodes. A guarded attribute grammar is **sound** if a closed configuration is reachable from any configuration $\Gamma$ reachable from the initial configuration $\Gamma_0(c) = \{X_0 = c\}$ associated with a case $c$.*

Let $\gamma$ denote the set of configurations reachable from the initial configuration of some case. We consider the finite sequences $(\Gamma_i)_{0<i\leq n}$ and the infinite sequences $(\Gamma_i)_{0<i}$ of configurations in $\gamma$ such that $\Gamma_i[\rangle\Gamma_{i+1}$. A finite and maximal sequence is said to be **terminal**, i.e., a terminal sequence leads to a configuration that enables no production. Soundness can the be rephrased by the two following conditions.

1. Every terminal sequence leads to a closed configuration.
2. Every configuration on an infinite sequence also belongs to some terminal sequence.

We define the soundness problem as follows : Given a GAG $\mathcal{G}$, is $\mathcal{G}$ sound? We define two reachability problems as follows: given a GAG $\mathcal{G}$ and a configuration $\Gamma$, is $\Gamma$ a reachable configuration of $\mathcal{G}$ ? Given a configuration $\Gamma'$, is $\Gamma$ reachable from $\Gamma'$ using productions of $\mathcal{G}$ ? These problems can unfortunately be proved undecidable by a simple encoding of Minsky machines.

THEOREM 4.3. *The soundness and reachability problems are undecidable in general for guarded attribute grammars.*

Despite this result, interesting subclasses of the model enjoy some monotony properties, and are well suited to distribution.

The principle of a *distribution* of a GAG on a set of locations is as follows: A GAG is distributed by partitionning its set of sorts according to *locations*. Each location maintains a local configuration, and subscribes to results provided by other locations. Productions are applied locally. When variables are given a value by a production, the location that computed this value sends messages to the locations that subscribed to this result. Messages are simply equations defining the value of a particular variable. Upon reception of a message, a subscriber updates its local configuration, that is update some of its variables, and may in turn produce new messages sent to subscribers of affected variables. A formal definition of the distribution framework is provided in the extended version. A GAG is said to be *distributable* if the above distribution scheme preserves its behaviour.

Recall that application of a production $P$ to a node $X$ requires a matching condition, that is construction of a pair of matchings $\sigma_{in}$ and $\sigma_{out}$. We say that a production $P$ is **triggered** in node $X$ if substitution $\sigma_{in}$ is defined, i.e., the patterns $p_i$ match the data $d_i$. A specification can be considered erroneous when a triggered transition is not enabled because the set of equations $\{y_j = u_j\sigma_{in} \mid 1 \leq j \leq m\}$ is cyclic.

Substitution $\sigma_{in}$, given by pattern matching, is monotonous w.r.t. incoming information and thus it causes no problem for a distributed implementation of a model. However substitution $\sigma_{out}$ is not monotonous: it may be undefined when information coming from a distant location makes the match of output attributes a cyclic set of equations.

DEFINITION 4.4. *A guarded attribute grammar is* ***input-enabled*** *if every production that is* triggered *in a reachable configuration is also* enabled.

For input-enabled GAGs, messages consumptions and application of productions commute (we refer interested reader to extended version for details). This property means in particular that distribution does not affect the global behavior of an input-enabled GAG.

THEOREM 4.5. *An input-enabled GAG is distributable.*

However, input-enabledness is a property of the whole set of reachable configurations, and is thus undecidable. Nevertheless one can find a decidable sufficient condition for input-enabledness (called *strong-acyclicity*), similar to the strong non-circularity of attribute grammars [2], and which can be checked by a simple fixed-point computation.

THEOREM 4.6. *Strong-acyclicity can be checked in polynomial time and a strongly-acyclic GAG is input-enabled (hence distributable).*

# 5. CONCLUSION

Guarded attribute grammar is a model of data-centric collaborative systems where emphasis is put on a simple mathematical syntax and semantics which can ease formal reasoning, a clear identification of stakeholder's decisions (the system is totally driven by user interactions), and an implicit lifecycle of artifacts which allows maximal concurrency and a straightforward distribution scheme. As already mentionned, our formalism is similar to logic formalisms such as Datalog or Prolog with, nonetheless, a distinction between inherited and synthesized positions of attributes. Workflow models such as BPMN have also been considered to describe case management systems. They allow for intuitive definition of concurrent threads, but usually abstract data from the descriptions. Distributed implementation has been considered for other artifact models, such as Guard-Stage-Milestone[1] [5]. However, it may require restructuring the original GSM schema and relies on locking protocols to ensure that the outcome of the global execution is preserved.

An artifact is a structured document with some active parts. Indeed, an open node is associated with a service call that implicitly describes the data to be further substituted to the node. This notion of *active documents* is close to the model of Active XML introduced by Abiteboul et al. [1] which consists of semi-structured documents with embedded service calls. Such an embedded service call is a query on another document, triggered when a corresponding guard is satisfied. The model of active documents can be distributed over a network of machines. This setting can be instancied in many ways, according to the formalism used for specifying the guards, the query language, and the class of documents. The model of guarded attribute grammars is close to this general scheme with some differences: First of all guards in GAG apply to a single node and its attributes, while guards in AXML are properties that can be checked on a complete document. The invocation of a service in AXML creates a temporary document (called the workspace) that is removed from the document when the service call returns. In GAGs, a service call adds new children to a node, and all computations performed for a service are preserved in the artifact. This provides a kind of monotony to artifacts, that can be an useful property for verification techniques.

In the future, we plan to design prototypes to analyze and implement a GAG description together with the required support tools (editor, parser, checker, simulators ...) to develop some representative case studies to check applicability and limitations of the model. In particular, in order to comply with real-life applications, we might have to use *non-autonomous GAG systems*, i.e., systems whose basic layer is given by a guarded attribute grammar but which is coupled with external facilities as making a query to a database or calling a web service. The main concern is then to evaluate the impact of these couplings on the distribution of the model (we should avoid distributed conflits).

[1] model for artifacts lifecycles used as a basis for the OMG standard *Case Management Model and Notation* (CMMN).

# 6. REFERENCES


[1] S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, and R. Weber. Active xml: A data-centric perspective on web services. In *BDA'02*, 2002.

[2] B. Courcelle and P. Franchi-Zannettacci. Attribute grammars and recursive program schemes i and ii. *Theor. Comput. Sci.*, 17:163–191 and 235–257, 1982.

[3] E. Damaggio, A. Deutsch, and V. Vianu. Artifact systems with data dependencies and arithmetic. *ACM Trans. Database Syst.*, 37(3):22, 2012.

[4] P. Deransart and J. Maluszynski. *A grammatical view of logic programming.* MIT Press, 1993.

[5] R. Eshuis, R. Hull, Y. Sun, and R. Vaculín. Splitting gsm schemas: A framework for outsourcing of declarative artifact systems. In *BPM*, volume 8094 of *LNCS*, pages 259–274. Springer, 2013.

[6] R. Hull. Artifact-centric business process models: Brief survey of research results and challenges. In *OTM 2008*, volume 5332 of *LNCS*, pages 1152–1163. Springer, 2008.

[7] D.E. Knuth. Semantics of context free languages. *Mathematical System Theory*, 2(2):127–145, 1968.

[8] A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Syst. J.*, 42:428–445, July 2003.

[9] J. Paakki. Attribute grammar paradigms - a high-level methodology in language implementation. *ACM Computing Surveys*, 27(2):196–255, 1995.