RC25298 (WAT1207-052) July 23, 2012
Computer Science

IBM Research Report

Parallel Processing for Business Artifacts with Declarative Lifecycles

Yutian Sun
Department of Computer Science
UC Santa Barbara
USA

Richard Hull, Roman Vaculín
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
USA

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Parallel Processing for Business Artifacts with Declarative Lifecycles (Full Version)

Yutian Sun[1] ⋆, Richard Hull[2] ⋆, and Roman Vaculín[2]

[1] Department of Computer Science, UC Santa Barbara, USA
[2] IBM T J Watson Research Center, USA

Abstract. The business artifact (a.k.a. business entity) approach to modeling and implementing business operations and processes, unlike the traditional process-oriented approach, is based on a holistic marriage of data and process, and enables a factoring of business operations based on key business-relevant conceptual entities that progress through the business. The recently introduced Guard-Stage-Milestone (GSM) artifact meta-model provides a hierarchical and declarative basis for specifying artifact lifecycles, and is substantially influencing OMG's emerging Case Management Modeling Notation standard. In GSM, milestones correspond to business-relevant operational objectives that an artifact instance might achieve, stages correspond to meaningful clusters of activity intended to achieve milestones, and guards correspond to events and/or conditions that can cause the launch of stages. The termination of stages and the achieving of milestones are governed by events and/or conditions. In previous papers one characterization of the operational semantics for GSM is based on the incremental, strictly serial firing of Event-Condition-Action (ECA) like rules. This paper develops a parallel algorithm equivalent to the sequential one in terms of externally observable characteristics. Optimizations and analysis for the parallel algorithm are discussed. This paper also introduces a simplification of the GSM meta-model that provides more flexibility and makes checking for well-formedness of GSM models simpler and more intuitive than in the preceding works on GSM.

1 Introduction

Business artifacts (a.k.a. business entities with lifecycles) are emerging as an important conceptual basis for modeling and implementing business processes and operations [12, 2, 9]. Unlike process-centric approaches, business artifacts enable a holistic marriage of the data- and process-centric perspectives, and permit a factoring of a scope of business that is often robust in the face of changes in the underlying business. A declarative approach to business artifacts, called Guard-Stage-Milestone (GSM) was recently introduced [10, 3, 9], and is substantially influencing OMG's emerging Case Management Modeling Notation [1]. Similar to Case Management, GSM is especially well-suited for business processes that include large unstructured parts where the process users apply much of their own judgement about what steps should be performed and in what order. Citations [3, 9] introduce the formal operational semantics for GSM and provide

⋆ This author supported in part by NSF grant IIS-0812578.

three equivalent formulations for it. One of these, which enables a direct implementation, is based on the incremental, serial firing of Event-Condition-Action (ECA) like rules, with each incoming event processed in strict sequence. This paper develops an optimized parallel algorithm for executing GSM processes with improved throughput and response time, and we also introduce some simplifications to the GSM meta-model.

A GSM model typically consists of several artifact types, where each artifact type corresponds to a class of key business-relevant conceptual entities that progress through the business. Each artifact type includes an *information model*, which holds all business-relevant information about an artifact instance as it progresses, and a *lifecycle model*, which represents the ways that the artifact instance might progress through the business. In GSM, *milestones* correspond to business-relevant operational objectives that an artifact instance might achieve, *stages* correspond to meaningful clusters of activity that are intended to achieve milestones, and *guards* control when stages can be opened for execution. The stages may be nested, and may be running in parallel. The processing is controlled by declarative expressions, called *sentries*, of the form "**on** ξ **if** φ" where ξ is an event expression and φ is a condition referring to the information model of the artifact instance (both ξ or φ are optional). Each guard is a sentry, and sentries are used to control when stages should open or close and when milestones get achieved or invalidated. The use of sentries provides a declarative basis for GSM, and nesting of stages provides abstraction and modularity. As described in [9], a GSM model typically involves several interrelated artifact types, and during execution there will be many artifact instances of each of these types. For ease of exposition, in the current paper we focus on the restricted case where there is one artifact type and one artifact instance. The results presented here naturally generalize to the multi-type/multi-instance context.

The most straightforward approach to operational semantics of GSM, called incremental semantics, is based on the incremental, strictly sequential firing of Event-Condition-Action (ECA) rules. In response to a single incoming event, the GSM system will fire all relevant rules (e.g., for opening/closing of stages or achieving/invalidating of milestones) until no more can be fired; after which the next incoming event can be processed. The Barcelona prototype [8] implements a variation of this sequential approach. We employed this implementation in various projects (e.g., [13]) and it turned out that oftentimes when many incoming events were occurring in a short time the strictly sequential processing of incoming events created a bottleneck in the system. This is primarily because often many ECA rules need to be processed as a response to one event, and the subsequent events have to wait till this processing is finished.

In general, such situations often occur in the context of collaborative problem solving [6] which can be naturally well supported by the GSM framework because it is event-driven, declarative, and has the ability to support business processes where the workers exercise considerable control over what tasks are performed and when. A typical pattern in collaborative problem solving involves a "shared artifact" [5] which serves as a coordination point of possibly many users and other processes. Therefore, shared artifacts — managed in GSM systems as artifact instances — are targets of possibly many concurrently incoming events (in particular for the web-scale [7] collaborative processes), and optimized implementations are needed to avoid bottlenecks.

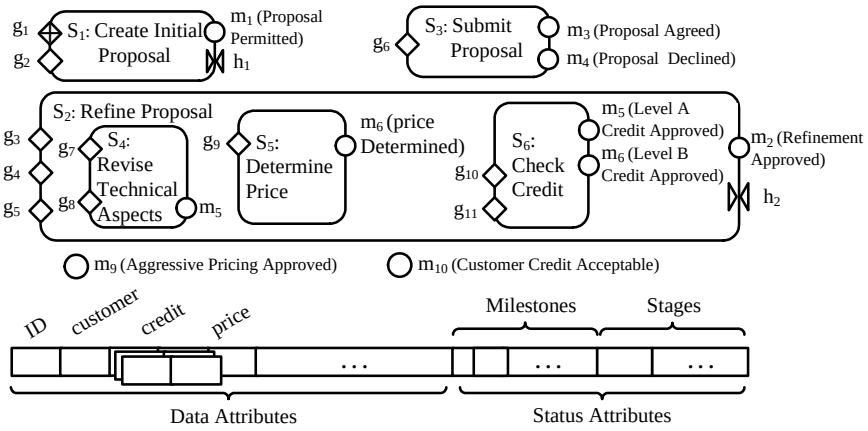

Fig. 1. Part of GSM model for *Proposal Creation* application

The major contribution of the present paper is a parallel algorithm for the execution of GSM models. In particular, we (a) use a graph analysis to "target" the set of sentries that might need to be tested, (b) introduce a pipelining technique so that the impact of multiple incoming events on an artifact instance can be processed in parallel, and (c) enable parallel execution of computation steps that stem from a single incoming event. We show that the developed parallel algorithm is equivalent in terms of the observable behavior with the sequential algorithm and we provide theoretical analysis which shows that in usual situations the parallel algorithm significantly improves efficiency.

A second contribution is to introduce a simplified GSM meta-model, called here '$\text{GSM}^{\text{kernel}}$' or simply 'GSM', that is more streamlined than the meta-model of [9, 3], called here '$\text{GSM}^{\text{classic}}$'. In $\text{GSM}^{\text{classic}}$ milestones are required to be attached to stages, which results in some intricacy in the definition of the well-formedness condition on GSM models. In $\text{GSM}^{\text{kernel}}$ milestones are by default separated from stages, and the definition of well-formedness is simplified. The style of milestones and stages used in $\text{GSM}^{\text{classic}}$ can be simulated using $\text{GSM}^{\text{kernel}}$ (see Example 3.4).

This paper is organized as follows. Section 2 provides a motivating example. The formal GSM meta-model and the operational semantics are presented in Section 3. Sections 4 and 5 focus on an optimized sequential algorithm and a parallel algorithm respectively. The proof of equivalence is presented in Section 6. Section 7 provides the optimizations and analysis of the parallel algorithm. Section 8 reviews related work, and Section 9 concludes the paper.

2 A Motivating Example

This section presents an example illustrating the key features of the GSM meta-model and the importance of developing a parallel algorithm for executing GSM processes.

The running example is taken from the domain of *Proposal Creation*, that is, creating proposals to provide large-scale IT solutions, which typically involve a combination of hardware, software, and services. Each proposal may take several months from

guards	**milestones**
g3: **on** +m1	m2 achiever: **on** refinedProposalReady()
g4: **on** +m4	m5 achiever: **on** revisedTechnicalReady()
g5: **on** resumeRefinement()	m5 invalidator: **on** +S4
g7: **on** +S2	m6 achiever: **on** priceDetermined(price)
g8: **on** offerManagerRequest()	m6 invalidator: **on** +S6
g9: **on** +m5	m7 achiever: **on** creditAppr(creditLevel) **if** creditLevel = "A"
g10: **on** +m6 and **if** not m7 & not m8	m8 achiever: **on** creditAppr(creditLevel) **if** creditLevel = "B"
g11: **on** +m6 and **if** price>500K & not m8	m9 achiever: **on** AggressivePricingExecApproval()
terminators	m9 invalidator: **on** AggressivePricingRescinded()
h1: **on** suspendCreation()	m10 achiever: **if** price <= 500K & (not m7 or not m8)
h2: **on** suspendRefinement()	m10 achiever: **if** price > 500K & m8
For each stage S with milestone m, each achiever for m is also a terminator for S	m10 invalidator: **on** +S6 **if** price > 500K & not m8

Fig. 2. Selected sentries for Example 2.1

launch to signing of a contract, and the solution itself may last several years. A large company may be creating several such proposals each month.

The proposal preparation work is highly collaborative, with each proposal involving 10's of participants that contribute their expertise into various steps. The proposal work tends to be bursty, with a flurry of activity in the days leading up to the release to the client of another version of the proposal. Whenever a worker inputs information into the shared workspace for a proposal, this may trigger several automated steps (e.g., sanity checks on what was just entered, updates to schedules, launching of automated routines such as recomputing component pricing, and/or launching of other human activities). Because of the bursty nature of the work, maximizing throughput is important. Because a step by some worker (e.g., giving an approval or submitting some information) may lead to an immediate follow-up step by the same worker or a close collaborator, minimizing response time is also important.

Example 2.1 Fig. 1 illustrates a small part of a GSM model that can support Proposal Creation, and Fig. 2 shows several sentries for that model. The information model of the primary business artifact is shown along the bottom of Fig. 1, and top layers of the lifecycle model are shown above. The three top-level phases of activity, depicted as GSM stages with rounded corner boxes, are *Create Initial Proposal* (S1), *Refine Proposal* (S2), and *Submit Proposal* (S3). Stage *Refine Proposal* is typically executed several times. Stages can be arranged hierarchically, as illustrated in *Refine Proposal*. (While not illustrated in this example, stages may be executing in parallel.)

GSM milestones, which correspond to business-relevant operational objectives, are shown as small circles. Several milestones are naturally associated with the completion of stages, e.g., *Refine Proposal* ends when an executive approves the refinement. Other milestones are free-standing. For example, *Customer Credit Acceptable* will become true if the proposal price is $\leqslant$ \$500K and the client has a Level A credit rating, and if the proposal price is > \$500K and the client has a Level B credit rating. Milestones can be tested as Boolean values by conditions anywhere in the GSM model.

Guards, which control when stages can be opened for execution, are shown as diamonds. A stage is opened if one of its guards is evaluated to true. Guard g_1 on *Create Initial Proposal* is a "bootstrapping" guard; when it is triggered a new artifact instance is created. In some cases it is convenient to have a *terminator* for a stage, shown us-

ing a bowtie. For example $h2$ on *Refine Proposal* is a terminator that can be triggered if a worker indicates that this stage should be suspended (e.g., because the client has stated that they are withdrawing their request). Unlike milestones, terminators cannot be tested as Booleans. ∎

We now briefly describe a representative scenario where fast response is important.

Example 2.2 Three representative substages are shown in *Refine Proposal*, namely *Revise Technical Aspects*, *Determine Price*, and *Check Credit*. In practice, *Revise Technical Aspects* would have a number of substages dealing with hardware, software, workforce, logistics, etc. *Determine Price* might involve several substages, and for this example we assume that they are all automated (e.g., for determining costs associated with various portions of the technical aspects, for computing shipping costs and taxes, for adding everything up). Only a subset of these substages would be relevant for a given execution of *Determine Price*. There may be other stages analogous to *Determine Price*, e.g., for determining risk, impact on branding, or impact on competitive positioning. The *Check Credit* stage might be triggered if a newly computed price is higher than the credit level already approved for the customer.

After a management-level worker approves a changed part of the *Revise Technical Aspects* stage, there will be processing to recompute the price, which will then lead to a feedback that either confirms that the client credit is still acceptable, or indicates that a higher-level Credit Check must be performed. It is desirable that this feedback be given to the worker within just a couple of seconds. ∎

As noted in the Introduction, in GSM$^{\text{classic}}$[3, 4, 9], all milestones are required to be associated with stages. In the kernel version of GSM used in this paper the milestones are free-standing by default. We view the attaching of milestones to stages, as illustrated in Fig. 1, as a notational convenience. (See also Example 3.4.)

3 Guard-Stage-Milestone Meta-model

This section presents the syntax and operational semantics of the GSM meta-model.

Definition: A *Guard-Stage-Milestone model*, or *GSM model*, Γ is a tuple (*Att*, *EType*, *Stg*, *Mst*, *Lcyc*), where

- *Att* is the set of *attribute names*, or simply, *attributes*. *Att* is partitioned into sets Att_{data} of *data attributes*, and Att_{status} of *status attributes*.
- *EType* is the set of *event types* (see below).
- *Stg* is the set of *stage names*, or simply, *stages*.
- *Mst* is the set of *milestone names*, or simply *milestones*.
- *Lcyc* is the lifecycle model, which will be defined below.

Sets *Att*, *EType*, *Stg*, *Mst* are pairwise disjoint. Att_{data} stores all the data relevant to the business process, while Att_{status} represent status values of milestones and stages. Domains of attributes are determined by a function *Dom*. For a data attribute A, $Dom(A)$ is assumed to include the undefined value $\perp$. For each $m \in Mst$, there is a Boolean *milestone status* attribute m in Att_{status}; this is *true* if the milestone has been achieved and not since invalidated, and *false* otherwise. For each $S \in Stg$, there is a Boolean *stage status* attribute S; this is *true* if the stage is open and *false* otherwise.

Artifact instances of GSM model Γ interact with the external environment by sending and receiving *typed external events* with event types defined in *EType*. An *event type* $E \in EType$ is of form $E_{\text{name}}(A_1, ..., A_n)$ where E_{name} is the event type name and A_i for $i \in [1..n]$ is in the Att_{data} of *Att*. An event instance e (or simply *event*) of an event type E is a tuple $E_{\text{name}}(A_1 : c_1, ..., A_n : c_n)$ where $c_i \in Dom(A_i)$, for $i \in [1..n]$. Tuple $(A_1 : c_1, ..., A_n : c_n)$ is called the *payload* of e. There are two types of external events: *incoming events* that are received from the external environment and *outgoing events* that are sent to the external environment.

The actual "work" of a GSM model is performed by *tasks*, which are contained in atomic stages. Tasks are invoked through message sending. There are three task types: (a) for generating 1-way messages (when invoked they wait for a "handshake" indicating success or failure); (b) generating 2-way service calls (when invoked they wait for the service call return from the called service or a time-out message); and (c) requesting the creation of a new artifact instance (which waits for a service call return with the ID of the newly created artifact instance). (Option (c) is not relevant to the current paper.) We note that computational tasks, including computations and assignments of attribute values are modeled using 2-way service calls; this allows for a separation between the repeated firing of sentries on the one hand and the impact of computing and assigning attribute values on the other.

The behavior a GSM artifact is defined by its lifecycle.

Definition: Given a GSM model Γ = (*Att*, *EType*, *Stg*, *Mst*, *Lcyc*), the lifecycle model *Lcyc* is a tuple (*Substages*, *Task*, *Submilestones*, *Guards*, *Terminators*, *Ach*, *Inv*) where

- *Substages* is a function from *Stg* to finite subsets of *Stg*. The relation $\{(S, S') \mid S' \in Substages(S)\}$ creates a forest, with roots called *top-level stages*, and leaves called *atomic stages*. Non-leaf nodes are called *composite stages*.
- *Task* is a function from the atomic stages in *Stg* to tasks.
- *Submilestones* defines a function from *Stg* to non-empty, finite subsets of *Mst*, s.t. $Submilestones(S) \cap Submilestones(S') = \emptyset$ for $S \neq S'$. (Top-level milestones will not be in the range of *Submilestones*.)
- *Guards* is a function from *Stg* to finite, non-empty sets of sentries (defined bellow).
- *Terminators* is a function from *Stg* to finite, non-empty sets of sentries.
- *Ach* is a function from *Mst* to finite, non-empty sets of sentries.
- *Inv* is a function from *Mst* to finite sets of sentries called *invalidating sentry* of m.

If $S \in Substages(S')$, then S is a *child-stage* of S' and S' is the *parent* of S. If $m \in Submilestones(S')$, then m is a *child-milestone* of S' and S' is the *parent* of m.

A *sentry* for GSM model Γ is an expression of form '**on** ξ **if** φ', '**on** ξ' or '**if** φ', where ξ is an *event expression* and φ is a well-formed formula that may or may not contain attributes from *Att*. An *event expression* is an expression ξ having one of the following forms: *Incoming event expression:* E, where $E \in EType$ (intuitively, it gets satisfied if an event of type E occurs); *Internal event expression:* For each milestone m this includes $+m$ and $-m$; and for each stage S this includes $+S$ and $-S$. Intuitively, $+m$ is triggered when m is achieved, $-m$ when m is invalidated, $+S$ when S is opened, and $-S$ when S is closed.

Operational Semantics: We now turn to the operational semantics for GSM models. These semantics were formally introduced in [3] (see also [4, 9]), and so the presentation here is brief. Citation [3] introduced three equivalent formulations of the operational semantics of GSM, called *incremental*, *fixpoint*, and *closed-form* (which is expressed in first-order logic). In the current paper we focus exclusively on the incremental formulation. This is based on responding to an incoming external event by repeated application of Event-Condition-Action (ECA) like rules until no further rules can be fired. The ECA-like rules are formed from the sentries of the GSM model.

Because negation is present in the ECA-like rules, some restrictions are placed on GSM models, and on the order of rule application. These restrictions, described below, are analogous to those found in stratified datalog and logic programming.

For the remainder of this section, unless otherwise specified, we use Γ to refer to a fixed GSM model (*Att*, *EType*, *Stg*, *Mst*, *Lcyc*).

Definition: A *pre-snapshot* of Γ is an assignment Σ that maps each attribute $A \in Att$ to an element of $Dom(A)$. A *snapshot* of Γ is a pre-snapshot that satisfies the following *GSM invariant*: for each pair of stages S, S', if $S \in Substages(S')$ and $\Sigma(S') = false$ then $\Sigma(S) = false$.

A (pre-)snapshot Σ is said to *satisfy* formula ψ, denoted as $\Sigma \models \psi$, if the ground formula ψ' evaluates to true, where ψ' is formed from ψ by replacing each occurrence of an attribute A by the value $\Sigma(A)$.

A fundamental notion in GSM is that of *Business Step* (*B-step*). These are conceptual atomic units of business-relevant processing, and correspond to the effect of incorporating one incoming external event into a snapshot of Γ. B-steps have the form of 4-tuples $(\Sigma, e, \Sigma', Gen)$, where Σ, Σ' are snapshots, e is an incoming external event, and *Gen* is a set of outgoing external events. As will be detailed below, under the incremental formulation this 4-tuple is a B-step if there is a sequence of pre-snapshots $\Sigma = \Sigma_0, \Sigma_1 = ImmEffect(e, \Sigma), \Sigma_2, \ldots, \Sigma_n = \Sigma'$ where, speaking intuitively, Σ_1 corresponds to the direct incorporation of event e into Σ, and Σ_{i+1} corresponds to the application of a sentry to Σ_i for $i \in [1..n-1]$. (For simplicity of exposition below, we also permit Σ_{i+1} to be identical to Σ_i for some i.) Further, *Gen* is the set of outgoing events caused by the opening of atomic stages during the sequence. In the formal model, computation of this sequence of pre-snapshots is assumed to happen in a single instant of time, and the set *Gen* of events is transmitted to the external environment in one batch immediately after Σ' is computed. We sometimes write a B-step as a triple (Σ, e, Σ') if *Gen* is understood from the context.

Example 3.1 In Fig. 1, suppose now S_1, S_3, S_5 and S_6 are closed and S_2 and S_4 are open. A new B-step can be triggered by receiving event *revisedTechnicalReady*. In this B-step, m_5 will be achieved and so stage S_4 will close. Since g_9 is $+m_5$, the B-step will also open stage S_5. After that no further sentries are applicable and the B-step ends. ∎

As detailed in [3, 4, 9], B-steps must satisfy two properties, called *Inertial* and *Toggle-Once*. Speaking intuitively, Inertial means that if Σ and Σ' differ on some status attribute, then there must be some PAC rule that justifies the change. Toggle-Once means that in a sequence as given above, a status attribute can change value at most

	Basis	Prerequisite	Antecedent	Consequent
Explicit rules				
PAC-1	Guard: if **on** ξ **if** φ is a guard of S. (Include term S' if S' is parent of S.)	$\neg S$	**on** ξ **if** $\varphi \wedge S'$	$+S$
PAC-2	Terminator: if **on** ξ **if** φ is an terminator of S.	S	**on** ξ **if** φ	$-S$
PAC-3	Milestone achiever: if m is a milestone and **on** ξ **if** φ is an achieving sentry for m. (Include term S' if S' is parent of m.)	$\neg m$	**on** ξ **if** $\varphi \wedge S'$	$+m$
PAC-4	Milestone invalidator: if m is a milestone and **on** ξ **if** φ is an invalidating sentry for m.	m	**on** ξ **if** φ	$-m$
Invariant preserving rules				
PAC-5	If S is a child stage of S'	S	**on** $-S'$	$-S$

Fig. 3. Prerequisite-Antecedent-Consequent Rules of a GSM Model

once. This corresponds to the intuition that everything business-relevant about a B-step should be observable by looking at the snapshots before and after the B-step

We now describe how the sentries of Γ are used to create the ECA-like rules, called here 'Prerequisite-Antecedent-Consequent' (PAC) rules. In typical GSM models that arise in practice, this family of rules will naturally satisfy the Toggle-Once property. However, in order to eliminate undesirable behavior from outlier GSM models, the PAC rules presented next include a mechanism that strictly enforces the Toggle-Once property.

Definition: A *Prerequisite-Antecedent-Consequent (PAC) rule* ρ, for GSM model Γ is a tuple (π, α, γ), where: (Prerequisite) π is a formula on attributes in Att; (Antecedent) α is a sentry based on attributes in Att, internal events over Att_{status}, and external event types $EType$; and (Consequent) γ is an internal event $\odot\sigma$, where $\odot \in \{+, -\}$ and $\sigma \in Att_{\text{status}}$.

Fig. 3 shows the template of PAC rules for a GSM model Γ. This family of PAC rule templates is simpler than the families presented in [3, 9], mainly because in the kernel GSM model milestones are completely separated from stages.

Consider an external event e of type E. Suppose there is a PAC rule $\rho = (\pi, \alpha, \gamma)$ for Γ where α has the form '**on** E' or '**on** E **if** φ' for external event type E. Suppose further that for some snapshot Σ of Γ, $\Sigma \models \pi$ and $\Sigma \models \varphi$ (if φ is part of α). Then e is *applicable* to Σ. In this case, the *immediate effect* of applying $e = E(A_1 : c_1, \ldots, A_n : c_m)$ to Σ, denoted $ImmEffect(\Sigma, e)$, is the result of replacing the value of A_j in Σ by c_j for each $j \in [1..m]$. (The impact of the PAC rule occurs in a subsequent step of the sequence $\Sigma_1, \ldots, \Sigma_n$.)

Suppose now that the sequence $\Sigma = \Sigma_0, \Sigma_1 = ImmEffect(\Sigma, e), \ldots \Sigma_k = \Sigma''$ has been constructed. A PAC rule $\rho = (\pi, \alpha, \gamma)$ is *applicable* to the pair (Σ, Σ'') if $\Sigma \models \pi$ and $\Sigma'' \models \alpha$. (For an event expression ξ occurring in α, if ξ is '**on** E' then it is satisfied if e has type E; if ξ is '**on** $+\sigma$' for status attribute σ then it is satisfied if $\Sigma \models \neg\sigma$ and $\Sigma' \models \sigma$; and analogously for '**on** $-\sigma$'.) Denote $apply(\Sigma, \Sigma'', \rho)$ as the result of flipping the sign of the value of the corresponding attribute in γ in snapshot Σ'' if ρ is applicable to (Σ, Σ''); otherwise (ρ is not applicable to (Σ, Σ'')), $apply(\Sigma, \Sigma'', \rho)$ is the same as Σ''.

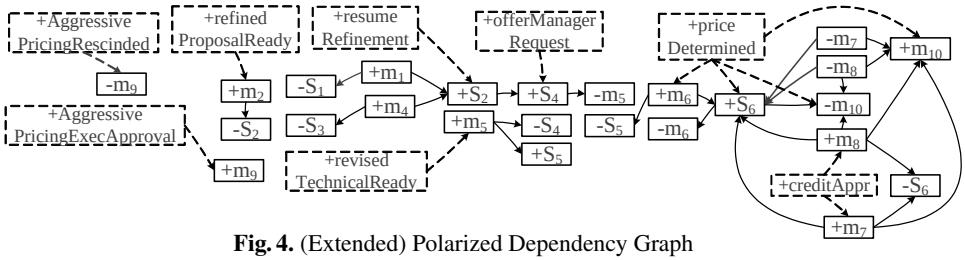

Fig. 4. (Extended) Polarized Dependency Graph

The following notion is needed to define the acyclicity condition that is imposed on GSM models.

Definition: The *polarized dependency graph* (*PDG*) of GSM model Γ, denoted $PDG(\Gamma)$ is a graph where the node set $\mathcal{V}$ includes: for each milestone m, nodes $+m$ and $-m$; and for each stage S, nodes $+S$ and $-S$. The edge set $\mathcal{E}$ is defined as follows. (Here "$\odot, \odot'$" are *polarities*, and range over $\{+, -\}$; σ, σ' are not necessarily distinct status attributes; and (π, α, γ) as a PAC rule in Γ_{PAC}.)

- If α includes in its triggering event ξ the internal event expression $\odot'\sigma'$, and γ is $\odot\sigma$, then include edge $(\odot'\sigma', \odot\sigma)$.
- If α includes in its condition the expression σ' and γ is $\odot\sigma$, then include edges $(+\sigma', \odot\sigma)$ and $(-\sigma', \odot\sigma)$.

Intuitively, an edge $(\odot\sigma, \odot'\sigma')$ is included in the PDG if PAC rules that might set σ according to $\odot\sigma$ should be considered before rules that might set σ' according to $\odot'\sigma'$.

Example 3.2 In Fig. 4, focus for now on the solid nodes and edges. These correspond to part of the PDG for the GSM model of Example 2.1. To illustrate, note that there is an edge from $+m_5$ to $+S_5$. This is because the guard g_9 on S_5 includes event $+m_5$. Also, both $+m_8$ and $-m_8$ point to m_{10} in the PDG, because m_8 is mentioned in the **if** part of an achiever of m_{10}. ∎

Definition: A GSM model Γ is *well-formed* if $PDG(\Gamma)$ is acyclic.

Given $PDG(\Gamma) = (\mathcal{V}, \mathcal{E})$ and PAC rule set Γ_{PAC} for GSM model Γ, define function *assoc* as a mapping from $\mathcal{V}$ to subsets of Γ_{PAC} so that $assoc(\odot\sigma) = \{(\pi, \alpha, \gamma) \in \Gamma_{PAC} \mid \gamma = \odot\sigma\}$. The notion of *B-step* can now be defined.

Definition: Given a GSM model Γ, snapshots Σ, Σ' of Γ, an incoming event e of Γ, and set of outgoing external events *Gen*, $(\Sigma, e, \Sigma', Gen)$ is a *business step* if there exists a topological sort $v_2, \ldots, v_n$ of the nodes of $PDG(\Gamma)$ and a sequence of pre-snapshots $\Sigma = \Sigma_0, \Sigma_1 = ImmEffect(\Sigma, e), ..., \Sigma_n$ of Γ such that $\Sigma_{i+1} = apply(\Sigma_0, \Sigma_i, assoc(v_i))$ for each $i \in [2, n-1]$ and $\Sigma_n = \Sigma'$.

A key result of [3] states that if Γ is well-formed and e is applicable to Σ, then there is exactly one snapshot Σ' such that (Σ, e, Σ') is a B-step. In particular, the construction of Σ' is independent of the topological sort used for rule application.

As briefly illustrated in the following example, the GSM meta-model used here can simulate the "GSM$^{\text{classic}}$" meta-model used in [3, 4, 9].

Example 3.3 Suppose GSM$^{\text{classic}}$ model Γ has a stage S with milestone m. To simulate this in a GSM model Γ', make m a free-standing milestone that has the same parent (if any) as S, and incorporate the following sentries.

1. For each achiever of m in Γ make it an achiever of m in Γ', and likewise for each invalidator.
2. For each guard γ of S in Γ that does not involve $\neg m$ as a top-level conjunct, add γ as an invalidator of m in Γ'.
3. For each achiever A of m create a terminator for S with the same sentry as A.

If S has no guards with $\neg m$ as a top level conjunct, then item (2) above can be replaced with "add '**on** $+S$' as an invalidator for m in Γ'". The current form of item (2) is related to the specialized form of PAC rule 4 in GSM$^{\text{classic}}$. ∎

The following example illustrates a subtlety that can arise from the Toggle-Once principle.

Example 3.4 (*Unstable outcome of B-step*) Consider a GSM model Γ that includes a top-level stage S and two top-level milestones m_1 and m_2. Suppose that S has guard '**if** m_2' and terminator '**if** m_1'; m_1 has achiever '**on E**', and that m_2 has achiever '**on** $+m_1$'. Suppose that Σ has S true and both m_1, m_2 false, and that an event e of type E is applicable to Σ. In the construction of the B-step (Σ, e, Σ') m_1 is achieved, which leads to invalidation of S and achieving of m_2. The PAC rule corresponding to the achiever '**if** m_2' of S is $(\neg S, m_2, +S)$; this is not fired in the construction of (Σ, e, Σ') because of the Toggle-Once principle, and more specifically because $\Sigma \not\models \neg S$. In this case, the snapshot Σ' is called *unstable*. ∎

A node $\odot\sigma$ in $PDG(\Gamma)$ is an *orphan node* after B-step (Σ, e, Σ') if Σ' is unstable because of some PAC rule with consequent $\odot\sigma$, i.e., if there is a PAC rule $(\pi, \alpha, \odot\sigma) \in \Gamma_{PAC}$, s.t. $\Sigma' \neq apply(\Sigma', \Sigma', (\pi, \alpha, \odot\sigma))$. A GSM model is called *orphan free* if there is no orphan node for any valid B-step (Σ, e, Σ') of Γ.

It is straightforward to develop syntactic conditions on GSM models that ensure they are orphan free (these focus on the existence of two directed paths from one status attribute to the two polarities of another status attribute). However, it remains open to find an exact syntactic characterization of orphan-free GSM models.

In practical terms, the presence of orphan nodes for a GSM model suggests that there are some counter-intuitive aspects to the design of the model. Thus, for the remainder of this paper we consider only orphan-free GSM models.

Observable behavior of an artifact: In operation, there is an interplay between an artifact and the environment, based on events being sent from each side to the other. In terms of observable behavior, the external environment has three modes in interaction with artifacts:

- Send external events to artifacts; or
- Receive outgoing events from artifacts; or
- Make queries against artifacts to obtain the current artifact snapshots.

Under the incremental formulation of the semantics, the outgoing events generated by a B-step caused by incoming event e should occur after e has been sent to the artifact, and before outgoing events generated by B-steps caused by incoming events subsequent to

e. Similarly, if a query Q is sent after event e and before event e', the answer received for Q should reflect the impact of the B-steps caused by e and all preceding events, and should not reflect the impact of B-steps resulting from e' and any succeeding events.

Example 3.5 Suppose e_1, e_2, q_1, e_3, q_2 is a sequence of incoming events and queries, where e_1, e_2 and e_3 are events while q_1 and q_2 are queries. Then the return result for q_1 should only show the snapshot with e_1 and e_2 incorporated but not e_3. However for q_2, all the effects of e_1, e_2 and e_3 should be reflected. ∎

The parallel algorithm developed in the next sections will have the same externally observable behavior as the incremental formulation of the GSM semantics.

4 Targeting Algorithm

The reason to propose the parallel algorithm for GSM is to solve the bottleneck that may occur when events are evaluated in a strictly sequential manner. The proposed parallel algorithm uses the following techniques. (1) *Targeting*: When processing a single B-step, it is sufficient to traverse only a subset of the PDG graph instead of the entire graph (introduced in this section); (2) *Pipelining*: B-steps can be evaluated in parallel (see Section 5); (3) *Parallelism within a single B-step*: Even within a B-step, some evaluations can be done in parallel (see Section 5).

While the techniques are inspired by traditional parallel processing, the novelty is in addressing the specific operational semantics of GSM and of B-steps. For example, suppose two B-steps are being evaluated in parallel by traversing the same PDG and by applying the appropriate PAC rules. Assume both B-steps are at some point modifying the value of a status attribute σ. If one B-step is going to visit node $+\sigma$ and the other one the node $-\sigma$, then the order in which the corresponding PAC rules of these two B-steps will be applied to these two nodes will affect the value of σ. Depending on the order (may be non-deterministic) the final value of σ may not be consistent with the snapshot generated by the sequential algorithm. To avoid such situations, the parallel algorithm incorporates various constraints to restrict the ordering.

We start by introducing the targeting algorithm which serves as a basis for the parallel algorithm. This approach reduces the number of PDG nodes that need to be evaluated in every B-step. We show the equivalence between the targeting algorithm and the sequential one. In order to reduce the number of PDG nodes to be visited during a B-step, it is necessary to extend the definition of PDG.

Definition: Let Γ be a GSM model and $PDG(\Gamma) = (\mathcal{V}', \mathcal{E}')$ be its PDG. The *extended polarized dependency graph* (*EPDG*) of Γ, denoted as $EPDG(\Gamma)$, is a graph $(\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ and $\mathcal{E}$ are defined as follows:

- For each node $v \in \mathcal{V}'$, add v to $\mathcal{V}$;
- For each event type E of Γ, add $+E$ to $\mathcal{V}$
- For each edge $(u, v) \in \mathcal{E}'$, add (u, v) to $\mathcal{E}$;
- If a guard g (or a terminator h) of a stage S contains a triggering event of type E, or a data attribute from the payload of E, then add edge $(+E, +S)$ (or $(+E, -S)$) to $\mathcal{E}$;

– If an achieving (or invalidating) sentry of milestone m contains a triggering event of type E, or a data attribute from the payload of E, then add edge $(+E, +m)$ (or $(+E, -m)$) to $\mathcal{E}$.

Example 4.1 Fig. 4 shows an example EPDG (with both solid and dashed nodes and edges) based on the GSM model in Fig. 1. ∎

Lemma 4.2 Given a GSM model Γ, if $PDG(\Gamma)$ is acyclic, then $EPDG(\Gamma)$ is acyclic.

Proof: Suppose there exists an cycle in the EPDG. Since the base PDG does not contain a cycle, the new formed cycle must contain an edge which is newly introduced. However all new edges in EPDG have source nodes which are of form $+E$ and for each $+E$, there is no incoming edge. Hence lead to a contradiction. ∎

The proof of Lemma 4.2 is based on the fact that adding a node without any incoming edges to a directed acyclic graph does not break acyclicity. Let $reachable(+E)$ denote a set of nodes in EPDG which are reachable from $+E$ (including $+E$). According to Lemma 4.2, the subgraph $reachable(+E)$ is also acyclic and a topological order in this subgraph can be obtained.

Given an EPDG node v, the *support* of v, denoted as $supp(v)$, is a set of attributes, s.t. an attribute A is in $supp(v)$ if and only if A is referenced by a PAC rule in $assoc(v)$.

In the remainder of this paper, we only consider well-formed and orphan-free GSM models.

Lemma 4.3 (Targeted impact lemma) Given an orphan-free and well-formed GSM model, a snapshot Σ, an incoming event e of type E, if Σ'' is some snapshot derived within the B-step (Σ, e, Σ') and ρ is a PAC rule associated to a node v, s.t. $v \notin reachable(+E)$, then $\Sigma'' = apply(\Sigma, \Sigma'', \rho)$.

Proof: Let v be an EPDG node; Σ, Σ' and Σ'' be snapshots; and e be an event of type E as given.

Since Γ is an orphan-free GSM model, before executing the B-step (Σ, e, Σ'), for each PAC rule (π, α, γ) of Γ, Σ is equal to $apply(\Sigma, \Sigma, (\pi, \alpha, \gamma))$. This implies that Σ (1) does not satisfy π, or (2) Σ does not satisfy α.

For case (1), suppose that PAC rule (π, α, γ) associated with v does not satisfy π. Then this rule is not applicable, that is, for each Σ'', Σ'' is equal to $apply(\Sigma, \Sigma'', (\pi, \alpha, \gamma))$.

For case (2), suppose that Σ satisfies π but not α. We now show by induction on a topological sort of EPDG that for each node v that is not reachable from $+E$, the PAC rules in $assoc(v)$ are not applicable. Assume v is the first node in the order consistent with the EPDG that is not in $reachable(+E)$. It is trivial to show that $supp(v) = \emptyset$. Therefore there is no referenced attributes in α or any triggering event in α. Hence, if Σ does not satisfy α, then Σ'' does not satisfy α, which means that $\Sigma'' = apply(\Sigma, \Sigma'', (\pi, \alpha, \gamma))$. Now let $k > 1$ and suppose that for each $k' < k$, if v' is the $(k')^{\text{th}}$ node in the topological sort and not in $reachable(+E)$, then no PAC rule associated with v' is applicable with Σ and Σ''. Now let v be the k^{th} node in the topological sort that is not in $reachable(+E)$. All the values of attributes in $supp(v)$ in Σ'' are the same as the ones in Σ according to the induction. Therefore, if Σ does not satisfy α, then Σ'' does not satisfy α, which implies that $\Sigma'' = apply(\Sigma, \Sigma'', (\pi, \alpha, \gamma))$. ∎

As an implication of Lemma 4.2 and 4.3, during the B-step triggered by an incoming event e of type E, a topological traversal can be performed using the subgraph reachable from $+E$ and only a corresponding subset of PAC rules needs to be applied.

Assuming an orphan-free, well-formed GSM model, the following "*targeting algorithm*" performs one B-step (Σ, e, Σ') for an incoming event e, and a snapshot Σ:

- For each edge node v in $reachable(+E)$ in topological order
  - If v is $+E$, then $\Sigma' := ImmEffect(e, \Sigma)$
  - Otherwise, for each PAC rule ρ associated with v, $\Sigma' := apply(\Sigma, \Sigma', \rho)$

Theorem 4.4 The targeting algorithm and the incremental algorithm share the same operational semantics.

Theorem 4.4 is a direct result from Lemma 4.2 and Lemma 4.3. If the multi-thread mechanism is used in the GSM engine, one optimization using the targeting algorithm can be obtained by evaluating two nodes that have no partial relationship in parallel. We use this idea in the parallel algorithm in Section 5 to handle multiple events.

Example 4.5 In Fig. 4, if event "priceDetermined" comes, instead of traversing the whole PDG, only nodes $+m_6$, $+S_6$, $-S_5$, $-m_6$, $+m_{10}$ and $-m_{10}$ need to be visited in topological order. Furthermore, since $-S_5$, $+m_{10}$, and $+S_6$ share no partial relationship, these three nodes can be evaluated in parallel. ∎

5 Parallelized Business Steps: Algorithm with Parallelism

In this section, we introduce a basic algorithm to parallelize B-steps and we show the equivalence between the parallel algorithm and the targeting algorithm. The basic algorithm has some drawbacks including some overheads. The purpose of the basic algorithm is to provide basic techniques of how to parallelize B-steps which are intuitive and easy for showing the equivalence. Building on this basic algorithm, we will introduce further optimizations in Section 6 which address the overheads.

The targeting algorithm in Section 4 provides the basis for parallelizing B-steps. Suppose two events of different types arrive at about the same time and the EPDG subgraphs of the two event types have no intersection in terms of nodes. Naturally, it is possible to traverse the two subgraphs and apply the corresponding PAC rules in parallel. On the other hand, if there are some nodes shared by the two graphs, the parallel algorithm needs to make sure that PAC rules associated with the shared nodes for the event that arrived first are applied before those associated with the later event. We assume that incoming events and the corresponding B-steps are labeled by increasing numbers (or logical timestamps) that correspond to the order in which events occurred (were added to the queue). The parallel algorithm will use these numbers to make sure that during the concurrent B-steps events ordering is not violated.

In the algorithms we use the following notation. Given a GSM model Γ, each node $v \in \mathcal{V}$ is associated with two sets denoted as $will\_visit(v)$ and $has\_visited(v)$, whose elements are positive integers. Given a status attribute σ, the *complement* of node $+\sigma$ (or $-\sigma$) is node $-\sigma$ (or $+\sigma$), denoted as $co(-\sigma)$ (or $co(+\sigma)$). Let $v \in \mathcal{V}$ be of form $\odot\sigma$, where $\odot$ ranges over $\{+, -\}$ and σ is a status attribute; function $attr(v)$ returns σ. Given a snapshot Σ and an attribute A, $\Sigma(A)$ represents the value of attribute A in Σ.

Algorithm 1 Artifact Initialization

Input: EPDG $EPDG(\Gamma) = (\mathcal{V}, \mathcal{E})$
Output: $\mathbf{\Sigma_\Gamma}$

1: Sets *will_visit* and *has_visited* are empty
2: $\mathbf{\Sigma_\Gamma}(\mathbf{0})$ is with all the data attribute values := $\perp$ and all the status attribute values := *false*
3: For all $i > 0$, $\mathbf{\Sigma_\Gamma}(i)$ is with all the attribute values := `Blank`
4: Integer $k := 0$
5: Launch scheduler (Alg. 2) with inputs $EPDG(\Gamma)$, *will_visit* and *has_visited*
6: **loop**
7: Wait for the next event e of type E
8: $k := k + 1$ {start to label}
9: **for each** $v \in reachable(+E)$ in topol. order **do**
10: $will\_visit(v) := will\_visit(v) \cup \{k\}$
11: **end for**
12: **end loop**

Algorithm 2 Basic scheduler

Input: $EPDG(\Gamma) = (\mathcal{V}, \mathcal{E})$, *will_visit*, *has_visited*

1: **loop**
2: **for each** node $u \in \mathcal{V}$ **do**
3: $k' := min(will\_visit(u))$ {check evaluation policy}
4: $V_{in} := \{v_{in} \mid (v_{in}, u) \in \mathcal{E}\}$
5: $N_{in} := \bigcup_{v_{in} \in \mathcal{V}_{in}}((will\_visit(v_{in}) \cup will\_visit(co(v_{in}))$
6: $k_{in} := min(N_{in})$
7: $k_{co} := min(will\_visit(co(u)))$
8: **if** $k' < k_{in}$ **and** $k' \leqslant k_{co}$ **then**
9: Create an evaluation thread (Alg. 3) with inputs k and u
10: **end if**
11: **end for**
12: **end loop**

Central to the algorithm is a data structure called *Parallel Evaluation Matrix* which serves for storing partial snapshots information.

Definition: Given a GSM model Γ, a *Parallel Evaluation Matrix* $\mathbf{\Sigma_\Gamma}$ is a 2-dimensional matrix, with each column corresponding to one attribute in the information model of Γ and with the rows labeled by integers corresponding to the different B-steps.

For a given attribute A in Γ and an integer k, $\mathbf{\Sigma_\Gamma}(A, k)$ represents the attribute A in the kth B-step, $\mathbf{\Sigma_\Gamma^{val}}(A, k)$ denotes its value, and $\mathbf{\Sigma_\Gamma}(k)$ denotes the snapshot generated by the kth B-step. k is called the *index* of $\mathbf{\Sigma_\Gamma}$.

Initialization of a Parallel Evaluation Matrix $\mathbf{\Sigma_\Gamma}$ starts from index 0 where $\mathbf{\Sigma_\Gamma}(0)$ is initialized with all data attributes set to undefined values ($\perp$) and all status attributes set to *false*. All other attributes in $\mathbf{\Sigma_\Gamma}(k)$ for $k > 0$ are assigned with value "`Blank`" to indicate that they are not filled by any values yet. For simplicity, the Parallel Evaluation Matrix stores all the generated snapshots. At the end of this section, an optimization is provided to reduce the storage overhead.

The parallel algorithm is divided into three parts: *Initialization* (Alg. 1) is the main entry point and it initializes the data structures; *Scheduler* (Alg. 2) is responsible for detecting which EPDG node is ready for evaluation of PAC rules of a particular B-step; and *evaluation thread* (Alg. 3) whose job is to apply PAC rules or an immediate effect of the node selected by the scheduler, and to update the corresponding snapshot for the particular B-step. For each artifact, one scheduler is running and multiple evaluation threads run concurrently. The following paragraphs give an overview of the algorithm.

For each incoming event, there are three high-level steps of how B-steps are processed in parallel: *labeling*, *evaluation*, and *removal*.

1. Labeling: In order to understand which node is available for immediate processing, it is necessary to label them first. Suppose the k^{th} event arrives which is of type E, then add k to $will\_visit(v)$ of each EPDG node that is reachable from $+E$ (line 10 in Alg. 1). This "*labeling policy*" is based on the result of the Targeted impact lemma (Lemma 4.3).

2. Evaluation: Alg. 2 implements the following *evaluation policy*: B-step k can update the value of attribute in v only if the referenced attribute values have all been already updated by the same or all the earlier B-steps.

Technically, first, the scheduler algorithm finds a node v, s.t. the smallest number k in $will\_visit(v)$ (1) is smaller than each number in $will\_visit(u)$ and $will\_visit(co(u))$, where u is a node and (u, v) is an edge; and (2) is no greater than each number in $will\_visit(co(v))$. Second, an evaluation thread is created to evaluate the found node v for B-step k. An evaluation thread in Alg. 3 applies PAC rules or an immediate effect of the node v for B-step k.

3. Removal: Once the evaluation thread has applied PAC rules on a node v for B-step k, k is moved from $will\_visit(v)$ to $has\_visited(v)$ which indicates that the B-step k has been visited v already. This removes the ordering restriction and will unblock the later B-steps, so that they can apply PAC rules on the shared nodes.

The following remark provides the intuition why in the evaluation policy, it is necessary to check a node and its complement.

Remark 5.1 (Checking with complements) We explain here some key intuitions underlying the evaluation policy implemented in Alg. 2. The basic idea is that a node v may be considered in connection with B-step k if, for each node u immediately preceding v in the order consistent with $EPDG(\Gamma)$ and for each j, $j \leqslant k$, B-step j either will not impact $attr(u)$ or has already impacted that attribute. This intuition is enforced by requiring that the least value in $will\_visit(u)$ is greater than k for each u preceding v. An added intricacy stems from the possible interaction between nodes of the form $+\sigma$ and $-\sigma$ for some status attribute σ. For example, suppose that $k' < k$, that B-step k' might cause $+\sigma$, and that B-step k might cause $-\sigma$. The possible impact of B-step k on σ should be computed only after the possible impact of B-step k' on σ has been computed. For this reason, before permitting the impact of B-step k on $+\sigma$ to be computed we require that the least value in $will\_visit(-\sigma)$ is no smaller than k.

Now we briefly discuss the idea of Alg. 2. In order to apply a PAC rule correctly, the data referenced by each PAC rule should be correct. More precisely, given a node v to be evaluated, the value of $attr(v)$ should have a correct value in the prerequisite and the antecedent; while each attribute in the support of v should be correct in the antecedent. Thus, the evaluation policy is to guarantee that before an evaluation thread can evaluate v for B-step k, $supp(v)$ and $attr(v)$ have the correct values by each time (1) picking the smallest number k from $will\_visited(v)$ to show that all the B-steps that are earlier than k have finished evaluation upon v; (2) comparing k with all the numbers in $will\_visited(u)$ and $will\_visited(co(u))$, where u has an outgoing edge directing to v, to show that all the attributes in $supp(v)$ have "stable" values up to and including B-step k; and (3) comparing k with the smallest number in $will\_visit(co(v))$ to show that the value of $attr(v)$ is "stable" up to and including B-step $k-1$.

The remainder of this subsection describes the intuition underlying Alg. 3.

Suppose a node v satisfies the evaluation policy for k. The algorithm has the effect of computing the effect of B-step k on $attr(v)$. There are two cases to consider.

If v is of type "$+E$" (line 1), where E is an event type, then the evaluation thread should compute the immediate effect of the k^{th} event and apply it to $\Sigma_\Gamma(k)$ (i.e. assign the payload of the k^{th} event to snapshot $\mathbf{\Sigma_\Gamma}(k)$).

Otherwise (line 4), v is of type $+\sigma$ or $-\sigma$ where σ is a status attribute. In this case, The evaluation thread should apply PAC rules to change the value of σ (if applicable). In order to apply a PAC rule, two snapshots are needed: one serves as the prerequisite and the other holds the latest values for $supp(v)$. The prerequisite should be the value that is obtained from B-step $k-1$. However, there is a possibility that the attribute $\mathbf{\Sigma_\Gamma}(\sigma, k-1)$ is still with value "`Blank`" because $k-1$ is not in $will\_visit(\cdot\sigma)$, i.e., B-step $k-1$ will not affect the value of σ. According to the Targeted impact lemma (Lemma **??**), σ will remain the same during the B-step $k-1$. With this idea, it is only necessary to find the greatest $i < k$ for which B-step k' was applicable to σ, and fill in all the blank values of of σ from index $i+1$ to $k-1$ (line 6 - 9). Then use $\mathbf{\Sigma_\Gamma}(k-1)$ as the prerequisite. For the snapshot that holds the latest values for $supp(v)$, assign all the latest non-blank attributes to a temporary snapshot Σ (line 10 - 14) and apply PAC rules (line 15 - 17) upon Σ with the prerequisite. Finally, assign the value of σ in Σ to $\mathbf{\Sigma_\Gamma^{val}}(\sigma, k)$ to make the update (line 18).

Snapshot maintenance: Processing of B-step k has finished if all the ks have been moved to the $has\_visited(\cdot)$ sets. In order to generate a correct snapshot and send correct outgoing events, it is necessary to fill in attributes whose values are still `Blank`. States *unfinished*, *finished* and *stable* are associated to each snapshot in $\mathbf{\Sigma_\Gamma}$ to help with filling the blank attributes. The following rules define transitions between these states.

- Initially, $\mathbf{\Sigma_\Gamma}(0)$ is *stable*; and for each $k > 0$, $\mathbf{\Sigma_\Gamma}(k)$ is *unfinished*
- For each incoming event with number k, if all the ks have been moved to $has\_visited(\cdot)$ in the EPDG, then $\mathbf{\Sigma_\Gamma}(k)$ is *finished*
- For each *finished* $\mathbf{\Sigma_\Gamma}(k)$, if $\mathbf{\Sigma_\Gamma}(k-1)$ is *stable*, then copy each attribute value that is `Blank` in $\mathbf{\Sigma_\Gamma}(k)$ from $\mathbf{\Sigma_\Gamma}(k-1)$ to $\mathbf{\Sigma_\Gamma}(k)$. $\mathbf{\Sigma_\Gamma}(k)$ becomes *stable*.

Algorithm 3 Evaluation Thread

Input: integer k, EPDG node v, *will_visit*, *has_visited*

1: **if** v is of form $+E$ where E is an event type **then**
2: e := the event with number k.
3: $\mathbf{\Sigma_\Gamma}(k) := ImmEffect(e, \mathbf{\Sigma_\Gamma}(k))$
4: **else**
5: Status attribute $\sigma := attr(v)$
6: i is the largest number where $\mathbf{\Sigma_\Gamma^{val}}(\sigma, i) \neq$ `Blank` **and** $i < k$
7: **for each** Integer idx from $i+1$ to $k-1$ **do**
8: $\mathbf{\Sigma_\Gamma^{val}}(\sigma, idx) := \mathbf{\Sigma_\Gamma^{val}}(\sigma, i)$
9: **end for**
10: Σ is a snapshot of Γ with each attribute having value `Blank`
11: **for each** attribute A in Σ **do**
12: j is the largest number where $\mathbf{\Sigma_\Gamma^{val}}(A, j) \neq$ `Blank` **and** $j <= k$
13: $\Sigma(A) := \mathbf{\Sigma_\Gamma^{val}}(A, j)$
14: **end for**
15: **for each** PAC rule $\rho \in assoc(v)$ **do**
16: $\Sigma := apply(\mathbf{\Sigma_\Gamma}(k-1), \Sigma, \rho)$
17: **end for**
18: $\mathbf{\Sigma_\Gamma^{val}}(\sigma, k) := \Sigma(\sigma)$
19: **end if**
20: $will\_visit(v) := will\_visit(v) - \{k\}$ {removal}
21: $has\_visited(v) := has\_visited(v) \cup \{k\}$

Whenever a snapshot is *stable*, all attributes are filled with non-blank values according to the above rules. Thus the artifact service center can send out outgoing events as soon as a snapshot becomes *stable*. If a query is sent from the outside environment to obtain the current snapshot of the system, the artifact service center should respond the query with the stable snapshot whose index is right before the sequence of the query.

Implementation remarks: In the real implementation, the size of the Parallel Evaluation Matrix can be optimized by employing a garbage collecting method. The matrix only needs to keep information about the latest *stable* snapshot. Similarly, Alg. 2 is simplified for clarity purposes and in the real implementation it can be embedded into evaluation threads for a lower overhead. The detailed approach is introduced in the next section.

6 Proof of Equivalence

This section provides the proof of correctness of the parallel algorithm.

For this development, we assume that the thread allocation scheme used for the algorithms is fair, in that each requested thread is eventually processed.

The next three lemmas show that in a single B-step, all the nodes that are reachable from $+E$, where E is the type of the current incoming event, will be evaluated in a manner that is consistent with the partial ordering implied by the corresponding EPDG.

Lemma 6.1 Given a well-formed and orphan-free GSM model together with its EPDG $(\mathcal{V}, \mathcal{E})$, suppose k is a positive integer and k is in $will\_visit(u)$ and $will\_visit(v)$ where $u, v \in \mathcal{V}$ and $(u, v) \in \mathcal{E}$. Then the k in $will\_visit(u)$ is removed earlier than the k in $will\_visit(v)$.

Proof: The proof can be directly observed from the evaluation policy. Since k can be moved to $has\_visited(v)$ only if when k is less than all the numbers in $will\_visit(u)$, if k is still in $will\_visit(u)$, node v cannot be evaluated. ∎

Lemma 6.2 Given a well-formed and orphan-free GSM model, suppose an event is labeled with number k, the algorithm will eventually delete k from each set $will\_visit(\cdot)$ that it was placed into.

Proof: Consider a sequence number n, and suppose that the associated incoming event is of type E.

When $n = 1$, it is smaller than any other sequence number. Thus, the evaluation policy for a node u will be satisfied as soon as B-step 1 has been applied to each node v preceding u in $EPDG(\Gamma)$. Because $EPDG(\Gamma)$ is acyclic, B-step 1 will eventually be applied to each node reachable from E.

Suppose now that $k > 1$, and that for each n, where $1 \leqslant n < k$, if n is placed into $will\_visit(u)$ for some u then it is eventually moved into $has\_visited(u)$.

Suppose now that the k^{th} B-step is triggered by an event of type E. A straightforward induction can now be used to show that for each node v in $reachable(+E)$, k will eventually be removed from $will\_visit(v)$. ∎

Lemma 6.2 shows that the parallel algorithm has no deadlock. Another important result implied by Lemma 4.3 and Lemma 6.2 is that for each B-step, each node that might be affected by the k^{th} B-step is eventually evaluated by the parallel algorithm (i.e. all the reachable nodes from the corresponding $+E$ node), since each reachable node is labeled with some integer and this integer will eventually be removed from the $will\_visit$ set at some time according to Lemma 6.2.

Lemma 6.3 For a sequence number k, if E is the type of the k^{th} incoming event, then in the parallel algorithm the evaluation of nodes for the k^{th} B-step will include all nodes in *reachable*$(+E)$, and in a manner consistent with the partial order on *reachable*$(+E)$.

Proof: Direct result based on Lemma 6.2 and Lemma 6.1. ∎

Lemma 6.4 Given a well-formed and orphan-free GSM model Γ together with its EPDG $(\mathcal{V}, \mathcal{E})$, suppose A is an attribute in Γ, k is a positive integer and Σ_k is the snapshot generated by B-step k using the targeting algorithm. For each non-blank value $\mathbf{\Sigma}^{\mathbf{val}}_{\mathbf{\Gamma}}(A, k)$, it is the same as $\Sigma_k(A)$.

Proof: If A is a data attribute and $\mathbf{\Sigma}^{\mathbf{val}}_{\mathbf{\Gamma}}(A, k)$ is non-blank, then $\mathbf{\Sigma}^{\mathbf{val}}_{\mathbf{\Gamma}}(A, k)$ is assigned based on the payload of the event k, and is not affected by the application of PAC rules. Thus, $\mathbf{\Sigma}^{\mathbf{val}}_{\mathbf{\Gamma}}(A, k) = \Sigma_k(A)$.

In the following, assume that A is a status attribute.

We proceed by induction on the sequence of B-steps. Suppose first that $k = 1$. As specified in Alg. 3, the PAC rules are applied with the values only in $\mathbf{\Sigma}_{\mathbf{\Gamma}}(0)$ and $\mathbf{\Sigma}_{\mathbf{\Gamma}}(1)$. Therefore, B-step 1 is executed as if it is the only B-step in the system. According to Lemma 6.3, the B-step will evaluate each reachable node in a manner consistent with the EPDG. It follows that for each non-blank attribute A in $\mathbf{\Sigma}_{\mathbf{\Gamma}}(A)$, $\mathbf{\Sigma}^{\mathbf{val}}_{\mathbf{\Gamma}}(A, 1) = \Sigma_1(A)$.

Suppose now that $k > 1$, and that for each n, where $1 \leqslant n < k$, for each attribute A in Γ, if $\mathbf{\Sigma}^{\mathbf{val}}_{\mathbf{\Gamma}}(A, n)$ is non-blank, then it is equal to $\Sigma_n(A)$.

In the following we show inductively that for each node $\odot A$ in *reachable*$(+E)$ where $\odot$ ranges over $\{+, -\}$, and each sequence number k, if $\mathbf{\Sigma}^{\mathbf{val}}_{\mathbf{\Gamma}}(A, k)$ is not blank , then it is equal to $\Sigma_k(A)$.

Suppose now that the k^{th} B-step is evaluating a node $\odot A$. The corresponding evaluation thread needs a prerequisite snapshot Σ' and a working snapshot Σ to trigger $apply(\Sigma', \Sigma, p)$. In the following context, we consider the construction of (1) Σ' and (2) Σ in Alg. 3.

(1) For Σ', according to Alg.3, it is equal to $\mathbf{\Sigma}_{\mathbf{\Gamma}}(i)$, where i is the largest number, s.t. $i < k$ and $\mathbf{\Sigma}^{\mathbf{val}}_{\mathbf{\Gamma}}(A, i)$ is not blank. Since A can only be evaluated by a thread when k is the smallest number in *will_visit*$(\odot A)$ and *will_visit*$(co(\odot A))$ according to the evaluation policy, no number between i and n will further change the value of A. According to the Targeted impact lemma (Lemma 4.3), if any B-step between i and k will not reach node $+A$ or $-A$, A will remain the same between B-step i and k. Thus, it is safe to propagate $\mathbf{\Sigma}^{\mathbf{val}}_{\mathbf{\Gamma}}(A, i)$ to each $\mathbf{\Sigma}^{\mathbf{val}}_{\mathbf{\Gamma}}(A, j)$, where j ranges from $i + 1$ to $k - 1$. Since $i < k$, $\mathbf{\Sigma}^{\mathbf{val}}_{\mathbf{\Gamma}}(A, i) = \mathbf{\Sigma}^{\mathbf{val}}_{\mathbf{\Gamma}}(A, k - 1) = \Sigma_{k-1}(A)$ by induction.

(2) For Σ, the value of each attribute A' is by copying from $\mathbf{\Sigma}^{\mathbf{val}}_{\mathbf{\Gamma}}(A', j)$ where j is the largest number, s.t. $j \leqslant k$ and $\mathbf{\Sigma}^{\mathbf{val}}_{\mathbf{\Gamma}}(A', j)$ is not blank. According to the evaluation policy, A can be evaluated by a thread only when each attribute $A' \in supp(\odot A)$ satisfies two cases: (i) if A' is a data attribute, k has been removed from the *will_visit*$(+E)$, where E is the corresponding event type; and (ii) if A' is a status attribute, k has been removed from the *will_visit*$(\odot' A')$ and *will_visit*$(co(\odot' A'))$, where $\odot'$ ranges over $\{+, -\}$. Thus, each attribute in $supp(\odot A)$ will not be changed before B-step k when the current evaluation thread is created. According to the induction, since all the non-blank values modified by B-step 1 to $k-1$ using the parallel algorithm are the same as the ones using the targeting algorithm, with the same referenced values in Σ' and Σ, $apply(\Sigma', \Sigma, p)$ can produce the same value for A as the one in $\Sigma_k(A)$.

In conclusion, if $\mathbf{\Sigma}_{\mathbf{\Gamma}}^{\mathbf{val}}(A,k)$ is non-blank, $\mathbf{\Sigma}_{\mathbf{\Gamma}}^{\mathbf{val}}(A,k)$ is equal to $\Sigma_k(A)$. ∎

The following theorem establishes that the parallel algorithm has an equivalent effect as the one under sequential execution.

Theorem 6.5 (Equivalence theorem) Suppose Γ is an orphan-free and well-formed GSM model. Given a sequence of incoming events, the snapshot and the outgoing events generated by each B-step k ($k > 0$) using the parallel algorithm are the same as the ones when $\mathbf{\Sigma}_{\mathbf{\Gamma}}(k)$ is stable using the targeting algorithm.

Proof: Consider each snapshot $\mathbf{\Sigma}_{\mathbf{\Gamma}}(n)$, where n is a non-negative integer.

When $n = 0$, snapshot $\mathbf{\Sigma}_{\mathbf{\Gamma}}(0)$ is stable and all the attributes in $\mathbf{\Sigma}_{\mathbf{\Gamma}}(0)$ are filled with the initial correct values according to the parallel algorithm.

Suppose for each n ranging from 1 to $k-1$, when $\mathbf{\Sigma}_{\mathbf{\Gamma}}(n)$ is stable, it is equal to the n^{th} snapshot generated by targeting algorithm.

Consider the situation when n is k. Once snapshot $\mathbf{\Sigma}_{\mathbf{\Gamma}}(k)$ has finished, all the non-blank values are correct according to Lemma 6.4. As a result from the Targeted impact lemma (Lemma 4.3), for those attributes that are blank, they should have the same values from the previous snapshot $\mathbf{\Sigma}_{\mathbf{\Gamma}}(k-1)$. Thus, $\mathbf{\Sigma}_{\mathbf{\Gamma}}(k)$ is equal to the k^{th} snapshot generated by targeting algorithm when that snapshot is stable.

Since for each sequence number k and each attribute A, $\mathbf{\Sigma}_{\mathbf{\Gamma}}^{\mathbf{val}}(A,k) = \Sigma_k(A)$, the activated stages under both algorithms for each B-step are the same. Hence, the same set of outgoing events will be sent by the targeting algorithm after the k^{th} B-step and by the parallel algorithm once the k^{th} snapshot becomes stable. Furthermore, the payloads of the outgoing events will be identical for the targeting algorithm and the parallel algorithm.

For each query that request the snapshot after the k^{th} B-step, since $\mathbf{\Sigma}_{\mathbf{\Gamma}}^{\mathbf{val}}(k)$ is equal to $\Sigma_k(A)$, the query can obtain the same snapshot generated by the parallel algorithm as if the system is running under the targeting algorithm. ∎

Theorem 6.5 guarantees that the parallel algorithm can correctly handle multiple incoming events in parallel and generate the same snapshots and outgoing events as if all incoming events are processed sequentially.

7 Optimizations and Analysis

In this section, we provide optimizations of the basic algorithm addressing the unnecessary overheads. Based on the optimized algorithm we provide an analysis to show the efficiency of the parallelization.

The main bottleneck of the algorithm in Section 5 is the scheduler (Alg. 2) because (a) there is only one single scheduler thread and (b) each node to be evaluated needs to be determined by the scheduler. To eliminate this bottleneck, we describe here how event scheduler function can be parallelized. We propose a new role, called "*event thread*", which will be created for every incoming event. The event thread is responsible for selecting the available nodes (similar to a scheduler) and for evaluating all the reachable nodes for its associated incoming event together with updating the corresponding

snapshot (similar to an evaluation scheduler). In essence, the improved parallel algorithm "projects" a centralized scheduler down to each event thread so that the event thread can evaluate a node as soon as it is available, and it works as follows:

1. When an incoming event arrives, the system allocates an event thread for it.
2. The event thread first atomically labels all the reachable nodes for the current event with the sequence number assigned (similar to lines 9 - 11 of Alg. 1).
3. Then the event thread traverses the reachable nodes for the current incoming event according to the topological order.
4. During the traversal, if the node to be visited satisfies the evaluation policy, then the event thread will evaluate the node and update the corresponding snapshot in the parallel evaluation matrix; otherwise, the thread will wait till the node becomes available.
5. When traversal finishes, the thread updates the blank attributes in the corresponding snapshot.

Analysis of the optimized algorithm

Compared with the sequential formulation of the operational semantics, an event thread in the optimized algorithm has the overheads because of (1) labeling all the reachable nodes atomically, (2) checking evaluation policy and (3) updating the blank attributes.

In order to make a fair comparison, we analyze the execution time used between the targeting algorithm and the optimized parallel algorithm.

Given an incoming event whose sequence number is i, suppose the number of reachable nodes for this event is n_i and the average time to apply the PAC rules in each node is $O(t)$, where t is the length of the PAC rules, i.e. the number of variables, operators, and constants used in the PAC rules. Furthermore, suppose among the n_i nodes, m_i ($m_i \leqslant n_i$) of them will have their values updated and the time to update one value is $O(1)$. Therefore, the total time for executing the i^{th} B-step is $T_i = O(tn_i + m_i) = O(tn_i)$.

Now, let us consider the parallel algorithm for the i^{th} incoming event (that will also visit n_i reachable nodes). In addition to the basic costs which are the same as in the case of the sequential (targeting) algorithm the overhead time needs to be considered. Suppose the time to label one node with a sequence number is $O(1)$, the time to check all the evaluation policies till nodes being available is $O(e_i)$, the number of the blank attributes is b_i ($b_i \leqslant n_i$), and the time to update one attribute is $O(1)$. Hence, the time for executing one event thread is $P_i = T_i + L_i + C_i$, where T_i is the cost for the basic evaluation (the same as the one for the targeting algorithm), $L_i = O(n_i)$ is the cost to label all the nodes, and $C_i = O(e_i + b_i)$ is total cost for checking the evaluation policy and updating blank attributes. Among the three costs, the time consumed by T_i and C_i can be parallelized, while L_i cannot be parallelized as the labeling is done atomically.

Suppose there are M events coming at almost the same time, the total time for the targeting algorithm is $O(\sum_{i=1}^{M} T_i) = O(t \sum_{i=1}^{M} n_i)$ and for the parallel algorithm is $O(\sum_{i=1}^{M} L_i + T) = O(\sum_{i=1}^{M} n_i + T)$, where T is the time for the parallel execution ($T_i + C_i$ part). As the i^{th} event thread may take $T_i + C_i = O(tn_i + e_i + b_i)$ time, from a pipelining perspective, T can be approximated as $T = \texttt{MAX}\{(\sum_{j=1}^{i} e_j) + (tn_i + b_i)\}$, where i is from

1 to M. This is because each e_i denotes the delay to set up a new pipeline, while $tn_i + b_i$ can be executed in parallel.

In order to simplify the analysis, assume each number of reachable nodes n_i is of the same size n, the number of the blank attributes for one B-step is also n (this assumption is made based on that the number of nodes is proportional to the number of status attributes), and each delay e_i is of the same length e. The cost for the targeting algorithm would be $O(t \sum_{i=1}^{M} n_i) = O(tnM)$ and the cost for the parallel algorithm would be $O(\sum_{i=1}^{M} n_i + \texttt{MAX}\{(\sum_{j=1}^{i} e_j) + (tn_i + b_i)\}) = O(nM + eM + tn + n) = O(nM + eM + tn)$. In the worst case, the delay e cannot be greater than tn, which can be regarded as the time for the previous event thread to evaluate all the reachable nodes. This means that the cost for the parallel algorithm (with overhead concerned) cannot be worse than the targeting algorithm in the worst case.

Now we consider (1) the situation where all the M incoming events are of the same type, or (2) the situation where the incoming events do not overlap in terms of the affected nodes. In these cases, the delay e is of size t, because the next event thread can evaluate a node as soon as the previous thread finishes the evaluation of the same node in situation 1 (or immediately after the event arrives in situation 2). Therefore, the cost for the parallel algorithm is $O(nM + eM + tn) = O(nM + tM + tn)$ which is far less than the cost $O(tnM)$ for the targeting algorithm.

8 Related Works

The GSM paradigm used here is based on the business artifact model originally introduced in [12, 11], but using a declarative basis [3, 9].

There is a strong relationship between the GSM model and Case Management [17, 16]; both approaches focus on conceptual entities that evolve over time, and support *ad hoc* styles of managing activities. The GSM framework provides a formal operational semantics [3, 9]. The core GSM constructs are being incorporated into the emerging OMG Case Management Modeling Notation standard [1], and there is ongoing work to adapt the GSM semantics to that context.

DecSerFlow [15] is inherently more declarative than GSM. GSM can be viewed as a reactive system that permits the use of a rich rules-based paradigm for determining, at any moment in time, what activities should be performed next.

There is a loose correspondence between the artifact approach and proclets [14]; both approaches factor a BPM application into "bite-size" pieces that interact through time. Proclets do not emphasize the data aspect, and support only message-based proclet interaction. In addition to supporting messages, GSM permits interaction of artifact instances through condition testing and internal event triggering.

Citation [5] presents a framework for supporting web-based collaborative business processes. They use a construct called *task artifact* to hold the complete collaboration state, and to help manage the response to incoming events. An interesting research question is to explore whether GSM could provide a useful approach for specifying possible lifecycles of task artifacts.

9 Conclusions

Business artifacts with Guard-Stage-Milestone (GSM) lifecycle models offer a flexible, declarative, and modular way to support collaborative business processes. The core constructs of GSM are being incorporated into the emerging OMG Case Management Modeling Notation standard. This paper presents an algorithm to support parallel execution of GSM processes, which can be especially useful in the context of highly collaborative and/or web-scale business processes. The paper also introduces a GSM meta-model that simplifies the previously published one.

This paper offers the opportunity for several follow-on investigations. An important next step is to implement and benchmark the parallel algorithm presented here, on both synthetic and "real" processes, to determine the practical gains in throughput and response time yielded. More broadly, it will be useful to examine existing and future application areas for GSM and Case Management to identify other ways to optimize overall performance.

References

1. BizAgi, Cordys, IBM, Oracle, SAP AG, Singularity (OMG Submitters) and Agile Enterprise Design, Stiftelsen SINTEF, TIBCO, Trisotech (Co-Authors). Proposal for: Case Management Modeling and Notation (CMMN) Specification 1.0, Feb. 2012. Document bmi/12-02-09, Object Management Group.
2. D. Cohn and R. Hull. Business artifacts: A data-centric approach to modeling business operations and processes. *IEEE Data Eng. Bull.*, 32(3):3–9, 2009.
3. E. Damaggio, R. Hull, and R. Vaculin. On the equivalence of incremental and fixpoint semantics for business entities with guard-stage-milestone lifecycles. In *Proceedings of the 9th International Conference on Business Process Management*, BPM '11, 2011.
4. E. Damaggio, R. Hull, and R. Vaculín. On the equivalence of incremental and fixpoint semantics for business entities with guard-stage-milestone lifecycles (full version), 2012. Journal version submitted for review; available upon request.
5. C. Dorn, R. N. Taylor, and S. Dustdar. Flexible social workflows: Collaborations as human architecture. *IEEE Internet Computing*, 16(2):72–77, 2012.
6. S. Dustdar. Caramba - a process-aware collaboration system supporting ad hoc and collaborative processes in virtual teams. *Distributed and Parallel Databases*, 15(1):45–66, 2004.
7. S. Dustdar and M. Gaedke. The social routing principle. *IEEE Internet Computing*, 15(4):80–83, 2011.
8. T. Heath, R. Vaculin, and R. Hull. Barcelona: A design and runtime environment for modeling and execution of artifact-centric business processes (demo). In *Proc. Intl. Conf. Business Process Mgmt. (BPM)*, 2011.
9. R. Hull et al. Business artifacts with guard-stage-milestone lifecycles: Managing artifact interactions with conditions and events. In *Proc. 5th ACM Intl. Conf. on Distributed Event-based Systems*, DEBS '11, pages 51–62, New York, NY, USA, 2011. ACM.
10. R. Hull et al. Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *Proc. 7th Intl. Workshop on Web Services and Formal Methods*, WS-FM'10, pages 1–24, Berlin, Heidelberg, 2011. Springer-Verlag.
11. S. Kumaran, P. Nandi, F. F. T. H. III, K. Bhaskaran, and R. Das. Adoc-oriented programming. In *SAINT*, pages 334–343, 2003.
12. A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Syst. J.*, 42:428–445, July 2003.

13. R. Vaculín et al. Declarative business artifact centric modeling of decision and knowledge intensive business processes. In *Proc. Intl. Conf. on Enterprise Distributed Objects Conference (EDOC)*, pages 151–160, 2011.
14. W. M. P. van der Aalst et al. Proclets: A framework for lightweight interacting workflow processes. *Int. J. Cooperative Inf. Syst.*, pages 443–481, 2001.
15. W. M. P. van der Aalst and M. Pesic. Decserflow: Towards a truly declarative service flow language. In *The Role of Business Processes in Service Oriented Architectures'06*, 2006.
16. W. M. P. van der Aalst and M. Weske. Case handling: a new paradigm for business process support. *Data Knowl. Eng.*, 53:129–162, May 2005.
17. W.-D. Zhu and et al. Advanced Case Management with IBM Case Manager. `http://www.redbooks.ibm.com/redpieces/abstracts/sg247929.html?Open`.