Parallel Processing for Business Artifacts with Declarative Lifecycles

Yutian Sun[1,2,⋆], 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 is based on a holistic marriage of data and process and enables a factoring of business operations based on key business-relevant conceptual entities. 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 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 [7,5]. 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 [2,5], and is substantially influencing OMG's emerging Case Management Modeling Notation [1]. Citations [2,5] introduce the operational semantics for GSM and provide 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 a simplified 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. 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

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

R. Meersman et al. (Eds.): OTM 2012, Part I, LNCS 7565, pp. 433–443, 2012.

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*. 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. For ease of exposition, in the current paper we focus on the restricted case where there is only 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. In practical projects using GSM [9] when many incoming events are occurring in a short time the incremental semantics may create a bottleneck in the system. Such situations often occur in the context of collaborative problem solving [4] which can be naturally well supported by GSM. A typical pattern in collaborative problem solving involves a "shared artifact" [3] which serves as a coordination point of possibly many users and other processes. Shared artifacts are targets of possibly many concurrently incoming events and optimized implementations are needed to avoid bottlenecks.

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 with the sequential algorithm. A second contribution is to introduce a simplified GSM meta-model that is more streamlined than the meta-model of [5,2]. In the simplified model, milestones are by default separated from stages, and the definition of well-formedness is simplified.

Section 2 provides a motivating example. The formal GSM meta-model and the operational semantics are presented in Section 3. Section 4 focuses on an optimized sequential algorithm and a parallel algorithm respectively. Section 5 reviews related work, and Section 6 concludes the paper. Due to space limitations the presentation here is terse; additional details and proofs of correctness are in the full paper [8].

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 *IT Proposal Creation*. The proposal preparation work is highly collaborative, with each proposal involving 10's of contributing participants and it tends to be bursty, with a flurry of activity in the days leading up to the proposal release. 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

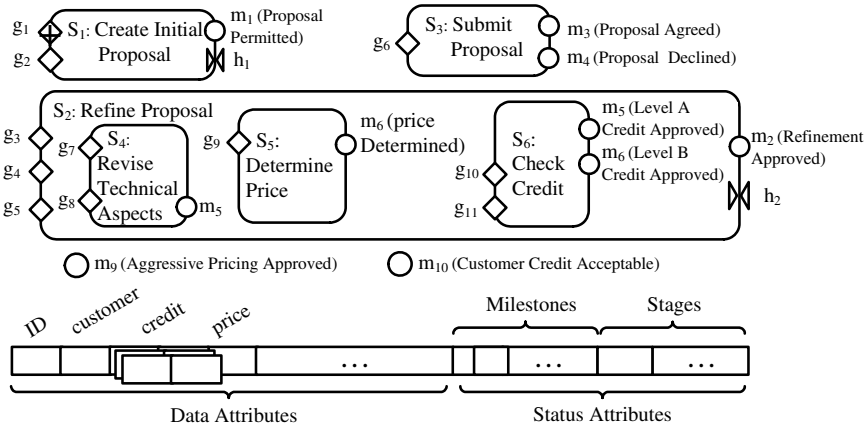

Fig. 1. Part of GSM model for *Proposal Creation* application

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

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 Figure 1 illustrates a small part of a GSM model that can support Proposal Creation, and Figure 2 shows several sentries for that model. The information model of the business artifact is shown along the bottom of Figure 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*. (Although not illustrated in this example, distinct stages may execute in parallel.)

GSM milestones are shown as 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. 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 using a bowtie. For example h_2 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, guards and 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. ∎

3 Guard-Stage-Milestone Meta-model

This section presents the syntax and operational semantics of the GSM meta-model. The focus is one GSM model that involves a single artifact type. Due to space limitations, the presentation is largely informal; formal definitions may be found in [8].

As illustrated in Section 2, a *Guard-Stage-Milestone model* (or *GSM model*) includes both an information model and a lifecycle model. This is typically denoted as a 5-tuple Γ = (*Att*, *EType*, *Stg*, *Mst*, *Lcyc*), where the components are, respectively, the set of *attributes* (partitioned into the set Att_{data} of *data attributes* and Att_{status} of *status attributes*; the set of (*incoming*) *event types*; the set of *stages*; the set of *milestones*; and the *lifecycle model* (which is defined below).

The *domain* of each data attribute A, denoted $Dom(A)$, is assumed to include the undefined value $\perp$. For each milestone m, there is a Boolean *milestone status* attribute also denoted by m in Att_{status}; this is *true* if the milestone has been achieved and not since invalidated, and *false* otherwise. For each stage S there is a Boolean *stage status* attribute also denoted by 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*. 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. An *event instance* (or simply, *event*) consists in an event type and a *payload*, specified as a family of attribute/value pairs. As a notational convenience, the attributes here are drawn from the set of data attributes of Γ. When an artifact instance incorporates an incoming event, the attributes mentioned in the payload are updated according to the values

The actual "work" of a GSM model is performed by *tasks*, which are contained in atomic stages. Tasks are invoked through message sending. The two types of task relevant for this paper can: (a) generate 1-way messages (when invoked they wait for a "handshake" indicating success or failure); (b) generate 2-way service calls (when invoked they wait for the service call return from the called service or a time-out message). (Upon completion of a 2-way service call generated by a task, an incoming message is received by the artifact instance, and the atomic stage associated with the task is closed.) In GSM, computational tasks (including assignments of attribute values) are modeled using 2-way service calls.

A *sentry* for GSM model Γ is an expression of form '**on** < *event expression* > **if** < *condition* >'; or '**on** < *event expression* >'; or '**if** < *condition* >'. The event expression may have 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. As illustrated in Section 2, the guards and terminators associated with stages are sentries. Also, milestones have associated achieving sentries and invalidating sentries.

The *lifecycle model* of a GSM model Γ = (*Att*, *EType*, *Stg*, *Mst*, *Lcyc*), is the tuple *Lcyc* = (*Substages*, *Task*, *Submilestones*, *Guards*, *Terminators*, *Ach*, *Inv*) where the components are, respectively a function that maps each stage to its family of substages (where the substage relationship forms a forest); a function that maps each atomic stage to the unique task that it contains; a function that maps a subset of the milestones to stages that they are contained in; a function that associates guards to stages; a function that associates terminators to stages; a function that associates achieving sentries to milestones; and a function that associates invalidating sentries to milestones.

A *snapshot* of a GSM model Γ is an assignment Σ that maps each attribute of Γ to a value in its domain (which includes $\perp$ for data attributes). Snapshots are required to satisfy the *GSM invariant*, namely that if a stage S is closed (i.e., if status attribute S is assigned the value false) then all of its child stages are also closed. In the following we also need the notion of *pre-snapshot*; this is an assignment Σ for Γ that might not satisfy the GSM invariant.

Citation [2] introduces 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.

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 event into a snapshot of GSM model 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. Under the incremental formulation, this 4-tuple is a B-step if there is a sequence of pre-snapshots $\Sigma = \Sigma_0, \Sigma_1 = \mathit{ImmEffect}(e, \Sigma), \Sigma_2, \ldots, \Sigma_n = \Sigma'$ where, speaking intuitively, Σ_1 corresponds to the direct incorporation (called "immediate effect", denoted as *ImmEffect*) of event e into Σ, and Σ_{i+1} corresponds to the application of a sentry to Σ_i for $i \in [2..n]$. (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 tasks that are launched 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. ∎

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 sentry that justifies the change. Toggle-Once means that in a sequence as given above, a status attribute can change value at most 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. The "prerequisite" component of these rules helps to enforce the Toggle Once property. (In most naturally arising GSM schemas, the "prerequisite" component is not needed.)

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 $\sigma \in Att_{\text{status}}$.

Returning to the incremental semantics, suppose that a partial sequence $\Sigma = \Sigma_0, \Sigma_1 = \mathit{ImmEffect}(e, \Sigma), \Sigma_2, \ldots, \Sigma_i$ has already been constructed. A PAC rule (π, α, γ) is *applicable* to (Σ, Σ_i) if $\Sigma \models \pi$ and $\Sigma_i \models \alpha$. In this case, Σ_{i+1} may be formed by modifying Σ_i according to the status change called for in γ.

A GSM model Γ is *well-formed* if the graph defined next is acyclic. This graph is also central to the parallelization developed in Section 4.

Definition: The *extended polarized dependency graph* (*EPDG*) of GSM model Γ, denoted $EPDG(\Gamma)$ is a graph where the node set $\mathcal{V}$ includes: for each event type E, nodes $+E$; 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}.)

	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 rule				
PAC-5	If S is a child stage of S'	S	**on** $-S'$	$-S$

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

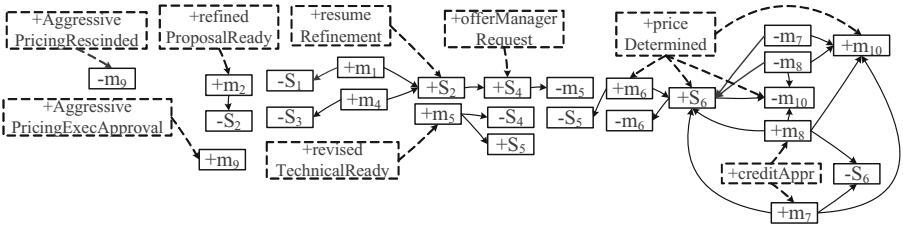

Fig. 4. (Extended) Polarized Dependency Graph

- 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 $(-\sigma', \odot\sigma)$.
- 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 include edge $(+E, +S)$ (or $(+E, -S)$).
- 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 include edge $(+E, +m)$ (or $(+E, -m)$).

Intuitively, an edge $(\odot\sigma, \odot'\sigma')$ is included in the EPDG if possible impacts on $\odot\sigma$ should be finalized before the PAC rules impacting $\odot'\sigma'$ are considered. We may also use the notion of "*PDG*" [8] (EPDG without $+E$ nodes, where E is an event type) in the remainder of this paper.

Fig. 4 shows to part of the EPDG for the GSM model of Example 2.1 (the solid nodes and edges form the "PDG"). Here, there is an edge from $+m_5$ to $+S_5$, because the guard g_9 on S_5 includes $+m_5$. Also, both $+m_8$ and $-m_8$ point to m_{10} in the EPDG, because m_8 is mentioned in the **if** part of an achiever of m_{10}.

When constructing a B-step using the incremental construction, the PAC rules are considered in an order based on a topological sort of the PDG A key result of [2] states that if Γ is well-formed and e is applicable to Σ, then there is exactly one snapshot Σ' (and one set Gen) such that $(\Sigma, e, \Sigma', Gen)$ is a B-step. In particular, the construction of Σ' is independent of the topological sort used for rule application.

As discussed in [8], in some corner cases after completion of a B-step $(\Sigma, e, \Sigma', Gen)$ some PAC rule might be applicable to Σ'. In this paper we consider only GSM models Γ for which this does not arise: such GSM models are called "*orphan-free*".

4 Parallelized Business Steps: Algorithm with Parallelism

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. (3) *Parallelism within a single B-step*: Even within a B-step, some evaluations can be done in parallel.

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.

Assuming an orphan-free and well-formed GSM model, the following "*targeting algorithm*" performs one B-step for an incoming event e of type E:

- For each reachable node v from $(+E)$ in topological order
  - If v is $+E$, then apply immediate effect with e.
  - Otherwise, apply each PAC rule associated with v.

Theorem 4.1 (Informal) The targeting algorithm and the incremental algorithm share the same operational semantics.

The proof of Theorem 4.1 is in [8]. If the multi-threaded 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 to handle multiple events.

Example 4.2 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. ∎

The targeting algorithm 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 subgraphs, the parallel algorithm needs to make sure that PAC rules associated with the shared nodes for the event that arrived earlier 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. The parallel algorithm will use these numbers to make sure that during the concurrent B-steps, the ordering of events is not violated.

In the algorithms, we use the following notation. Given a GSM model Γ and its EPDG $(\mathcal{V}, \mathcal{E})$, 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 $co(+\sigma)$).

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 v that is reachable from $+E$. This "*labeling policy*" is based on the result of the targeting algorithm.

2. Evaluation: Once the numbers are labeled for the k^{th} event, the B-step k can start to evaluate its corresponding reachable nodes similar to targeting algorithm (by applying the PAC rules or the immediate effect). A B-step k can evaluate a node v, if v satisfies the following *evaluation policy*: (1) k is the smallest number in *will_visit*(v); and (2) k 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 (3) k is no greater than each number in *will_visit*($co(v)$).

3. Removal: Once B-step k applies the PAC rules or the immediate effect on a node v, k is moved from *will_visit*(v) to *has_visited*(v) which indicates that the B-step k has 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.

Theorem 4.3 (Equivalence theorem, informal) . The parallel algorithm and the targeting algorithm are equivalent in terms of the observable behavior.

The formal statement and the proof of Theorem 4.3 is in [8]. Theorem 4.3 guarantees that the parallel algorithm can correctly handle multiple incoming events in parallel and generate the same snapshots, outgoing events, and query answers as if all incoming events and queries are processed sequentially.

In [8], an optimized algorithm is provided. Based on the optimized version, it can be shown that the complexity in the worst case is no worse than the targeting algorithm; while much better than targeting algorithm in terms of order in the best case.

5 Related Works

The GSM paradigm used here is based on the business artifact model originally introduced in [7,6], but using a declarative basis [2,5].

There is a strong relationship between the GSM model and Case Management [13,12]; 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 [2,5]. 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 [11] 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 [10]; 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 [3] 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.

6 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.

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, Document bmi/12-02-09, Object Management Group (February 2012)
2. Damaggio, E., Hull, R., Vaculín, R.: On the Equivalence of Incremental and Fixpoint Semantics for Business Artifacts with Guard-Stage-Milestone Lifecycles. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 396–412. Springer, Heidelberg (2011)
3. Dorn, C., Taylor, R.N., Dustdar, S.: Flexible social workflows: Collaborations as human architecture. IEEE Internet Computing 16(2), 72–77 (2012)
4. Dustdar, S.: Caramba - a process-aware collaboration system supporting ad hoc and collaborative processes in virtual teams. Distributed and Parallel Databases 15(1), 45–66 (2004)
5. Hull, R., 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 2011, pp. 51–62. ACM, New York (2011)
6. Kumaran, S., Nandi, P., Heath, T., Bhaskaran, K., Das, R.: Adoc-oriented programming. In: SAINT, pp. 334–343 (2003)
7. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Syst. J. 42, 428–445 (2003)
8. Sun, Y., Hull, R., Vaculín, R.: Parallel processing for business artifacts with declarative lifecycles (full version). IBM internal technical report, available on request (2012)
9. Vaculín, R., et al.: Declarative business artifact centric modeling of decision and knowledge intensive business processes. In: Proc. Intl. Conf. on Enterprise Distributed Objects Conference (EDOC), pp. 151–160 (2011)

10. van der Aalst, W.M.P., et al.: Proclets: A framework for lightweight interacting workflow processes. Int. J. Cooperative Inf. Syst., 443–481 (2001)
11. van der Aalst, W.M.P., Pesic, M.: Decserflow: Towards a truly declarative service flow language. In: The Role of Business Processes in Service Oriented Architectures 2006 (2006)
12. van der Aalst, W.M.P., Weske, M.: Case handling: a new paradigm for business process support. Data Knowl. Eng. 53, 129–162 (2005)
13. Zhu, W.-D., et al.: Advanced Case Management with IBM Case Manager, http://www.redbooks.ibm.com/redpieces/abstracts/sg247929.html?Open