On the Equivalence of Incremental and Fixpoint Semantics for Business Artifacts with Guard-Stage-Milestone Lifecycles

Elio Damaggio[a], Richard Hull[b,1], Roman Vaculín[b]

[a] *University of California, San Diego*
[b] *IBM T.J. Watson Research Center, New York*

Abstract

Business artifacts (or simply, artifacts) are used to model conceptual entities that are central to guiding the operations of a business, and whose content changes as they move through those operations. The recently introduced Guard-Stage-Milestone (GSM) meta-model for artifact lifecycles is declarative in nature, and allows concurrent execution of long-running (possibly human-executed) activities. Modularity is incorporated through the use of hierarchical clustering of activities. Milestones are used to track whether key business-relevant operational objectives have been achieved by a business artifact. The GSM operational semantics is based on a variant of Event-Condition-Action (ECA) rules, which are used to control the start and termination of individual and composite activities, and the recording of milestone status. This paper introduces, in an abstract setting, three different and provably equivalent formulations of the GSM operational semantics. The semantics is specified in terms of how a single external event is incorporated into the current "snapshot" (i.e. full description) of a running execution of an artifact model. The "incremental" formulation corresponds to the sequential application of the ECA-like rules in response to the event; the "fixpoint" formulation characterizes the mathematical properties of pairs of snapshots corresponding to the full impact of incorporating the event; and the "closed-form" formulation captures the fixpoint one in terms of first-order logic. The paper introduces a formally specified well-formedness condition on GSM models that guarantees the equivalence of the three formulations while permitting commonly arising patterns for using GSM constructs to model business operations.

Keywords: Business Artifacts; Business Entities with Lifecycles; Business Process Management; Case Management; Declarative Semantics.

Email addresses: `elio@cs.ucsd.edu` (Elio Damaggio), `hull@us.ibm.com` (Richard Hull), `vaculin@us.ibm.com` (Roman Vaculín)

[1]This author supported in part by NSF grant IIS-0812578 and by the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement number 257593 (ACSI).

1. Introduction

There is increasing interest in frameworks for specifying and deploying business operations and processes that combine both data and process as first-class citizens. One such approach is called Business Artifacts[2] (or simply “artifacts”), and has been studied by a team at IBM Resarch for several years [35, 25, 39, 6, 9, 34, 11]. Artifacts are key conceptual entities that are central to the operation of part of a business and that change as they move through the business’s operations. An artifact type includes both an *information model* that uses attribute/value pairs to capture, in either materialized or virtual form, all of the business-relevant data about artifacts of that type, and a *lifecycle model*, that specifies the possible ways that an artifact of this type might progress through the business, and the ways that it will respond to events and invoke external services, including human activities. The recently introduced [22] Guard-Stage-Milestone (GSM) meta-model[3] for Business Artifact lifecycles (abbreviated simply as “GSM”) provides a substantially declarative approach for specifying artifact lifecycles that supports parallelism and modularity, with an operational semantics based on a variant of Event-Condition-Action (ECA) rules.

As described in [22], the core motivation leading to GSM has been to create a meta-model for specifying business operations and processes that:

1. Will help business-level stakeholders to gain insight into and understanding of their business operations;
2. Is centered around intuitively natural constructs that correspond closely to how business-level stakeholders think about the operations of their business;
3. Can provide a high-level, abstract view of the operations, and gracefully incorporate enough detail to be executable;
4. Can support a spectrum of styles for specifying business operations and processes, from the highly “prescriptive” (as found in, e.g., BPMN) to the highly “descriptive” (as found in Adaptive Case Management systems); and
5. Can serve as the target into which intuitive, informal, and imprecise specifications of the business operations (e.g., in terms of “business scenarios”) can be mapped.

A fundamental research challenge underlying the development of GSM has been to find a meta-model that on the one hand incorporates the business-level constructs in an intuitive and flexible manner, and on the other hand supports a precise semantics that can support both implementation and mathematical investigation. The current paper presents the core constructs of GSM and the formal foundations for the GSM operational semantics. This includes specification of a well-formedness condition on GSM models, and proving the equivalence of three different formulations of the semantics, with each having its own intrinsic value. The well-formedness condition, while somewhat intricate, was chosen so that most commonly arising patterns for using the

[2]In some publications the term “Business Entity (with Lifecycle)” is used in place of “Business Artifact”.

[3]Following the tradition of UML and related frameworks, we use here the terms ‘meta-model’ and ‘model’ for concepts that the database and workflow research literature refer to as ‘model’ and ‘schema’, respectively.

GSM constructs can be specified using well-formed GSM models. We expect that this framework and approach can be adapted to variants of the GSM meta-model.

There are four key elements in the GSM meta-model: (a) *Information Model* for business artifacts, as in all variations of the artifact paradigm; (b) *Milestones*, which correspond to business-relevant operational objectives, and are achieved (and possibly invalidated) based on triggering events and/or conditions over the information models of active artifact instances; (c) *Stages*, which correspond to clusters of activity intended to achieve milestones; and (d) *Guards*, which control when stages are activated, and as with milestones are controlled through triggering events and/or conditions. Multiple stages of an artifact instance may be active (or “open”) at the same time, which enables the modeling of parallel activity such as when two human performers are simultaneously conducting different tasks in connection with the same artifact instance. Hierarchical structuring of the stages supports a rich form of modularity.

The GSM meta-model is an outgrowth of several years of practical experience with earlier artifact-centric meta-models, combined with a desire to provide a more flexible and goal-driven approach than offered by the previous meta-models. For example, this led to the use of a largely declarative paradigm, and to the use of milestones based on ECA-like rules as a central building block for artifact lifecycles.

The operational semantics for GSM is specified in terms of how a single “incoming event” is incorporated into the current “snapshot” (i.e., description of all relevant aspects at a given moment of time) of a GSM system. This semantics is based on a variant of the Event-Condition-Action (ECA) rules paradigm, and is centered around *GSM Business steps* (or *B-steps*), which focuses on the full impact of incorporating the incoming event. In particular, the focus is on what milestones are achieved or invalidated, and what stages become active or inactive, as a result of this incoming event. Changes in milestone and/or stage status are treated as internal “status events”, and can trigger further status changes in the snapshot. Intuitively, a B-step corresponds to the smallest unit of business-relevant change that can occur to a GSM system.

The semantics for B-steps has three equivalent formulations, each with their own value. These are:

Incremental: This corresponds roughly to the incremental application of the ECA-like rules, provides an intuitive way to describe the operational semantics of a GSM model, and provides a natural, direct approach for implementation.

Fixpoint: This provides a concise “top-down” description of the effect of a single incoming event on an artifact snapshot. It is useful for developing alternative implementations for GSM, and optimizations of them; something especially important if highly scalable, distributed implementations are to be created.

Closed-form: This straight-forward rewriting of the fixpoint formulation provides a characterization of snapshots and the effects of incoming events using a first-order logic formula. This permits the application of previously developed verification techniques to the GSM context. (The previous work, [7, 15, 12], assumed that tasks were performed in sequence, whereas in GSM tasks and other aspects may be running in parallel.)

This paper formally defines these three formulations of the semantics, and shows that they are equivalent for GSM models that satisfy the well-formedness condition. The development is in some ways reminiscent of logic programming [29], and the well-formedness condition, which is based on an acyclicity condition, can be viewed as providing a kind of stratification on the ECA-like rules. For ease of presentation, and without fundamentally compromising the applicability of the results, this paper uses a common restriction that puts the focus on a single artifact instance of a single artifact type. Citation [23] presents an extension of the meta-model described here that supports multiple artifact types, multiple artifact instances, and structured attribute values. (See also [24].)

While this paper focuses on the foundations of the GSM meta-model, it is also being studied from the practical perspective. A prototype engine, called Barcelona, is being developed to support experiments and implementations using GSM (discussed briefly in Section 6). Importantly, the incremental semantics introduced in this paper serves as a basis for implementation of Barcelona GSM execution engine. We are currently working on two variations of GSM implementation, one [40] taking advantage of highly parallel multi-threaded architectures for allowing very efficient implementation of GSM execution engine, and the other one being built on top of a distributed publish-subscribe platform. These two alternative approaches are taking advantage of the fixpoint formulation to make sure that they fully satisfy the GSM operational semantics. Finally, there is another ongoing thread of activities focused on formal verification of GSM models that leverage results of the closed-form formulation.

As briefly discussed in Section 7, there is a strong connection between GSM and the area of Adaptive Case Management [41], and with the emerging OMG standard [8]. In particular, the consortium leading the OMG effort has adopted the core GSM constructs of stages, milestones, guards and sentries, and work is underway to adapt the GSM semantics to that meta-model.

This paper is an extension of publication [13] and it extends it in several ways. The informal description of the GSM meta-model is considerably expanded, including both a listing of key GSM concepts and an extended example. The formal development of the GSM meta-model is expanded and presented in a comprehensive manner. The discussion of the condition for well-formed GSM models is expanded, and includes examples of what can go wrong with non-well-founded models. Finally, detailed proofs of the key results are included.

Organizationally, Section 2 provides an informal introduction to GSM, including both a listing of key concepts and an extended example. Section 3 presents a series of formal definitions for GSM, leading up to the key notion of GSM model. Section 4 presents the notion of B-steps in an informal way, and introduces some additional formalism, including the variant of ECA rules used, and the well-formedness condition on GSM models. Section 5 presents the three formulations of B-step and proves their equivalence; B-steps form the core of the GSM operational semantics. Section 6 briefly discusses GSM implementation; how the equivalence theorem can be used in connection with verification; and it discusses some aspects of sequences of B-steps. Section 7 overviews related work and Section 8 concludes the paper.

2. Informal Introduction to GSM

This section provides an informal and intuitive introduction to the GSM meta-model used for this paper. Listings of key constructs in the GSM meta-model, and of key concepts for the operational semantics, are provided. Then an example GSM model is described in detail.

2.1. Key constructs in GSM

The four key elements of the GSM meta-model are *Information Model*, *Milestones*, *Stages*, and *Guards*. Here we present informal, intuitive descriptions of these central notions of GSM, as well as several related constructs, including artifact instance, task, environment, events, etc. These are formalized primarily in Section 3. In the listing below, the most important constructs are indicated in ***bold italics***, and other constructs indicated in **bold**. The conceptual underpinnings of the GSM meta-model described in this listing were developed collectively by the several authors of [22, 23].

Artifact Instance: Artifact Instance corresponds intuitively to a single business-relevant conceptual entity (of a particular artifact type) that progresses through some business operations. By way of illustration, we note that in the financial application described in [9] one of the artifact types corresponds to the notion of a financial *Deal*, i.e., a loan from one party to another secured by some collateral. An artifact instance of a Deal can be in existence for several years, and progresses through activities involving checking on the borrower and the collateral, completing negotiations and a contract, and managing the periodic payments until termination of the contract.

Information model: Information model is an integrated view of all business-relevant information about an artifact instance as it moves through the business operations. This has two components, the *data attributes* and the *status attributes*. Data attributes hold data about the business itself and how it is being affected by the artifact instance. Status attributes hold "control information", that is, information about the progress of the artifact instance as it moves through the business operations. This information is indicated by the current status of milestones and stages in that instance .

Task: Task corresponds to a unit of business-relevant work that is to be performed by an outside agent (either human or machine). Tasks are invoked by artifact instances. When a task is invoked, the artifact instance provides input data from its information model, and when the task terminates the task output data is written into the artifact instance information model.

Environment: Corresponds to environment within which artifact instances exist. In the current paper the environment hosts the task executions that are invoked by artifact instances.

Event: In GSM there are three categories of event.

- *Outgoing Event:* Outgoing events are events sent from artifact instances to the environment. There is one kind of outgoing event, namely *task invocation event*, which correspond to when an artifact instance invokes an occurrence of a task.
- *Incoming Event:* Incoming events include events that can be sent from the environment to artifact instances. There are two kinds of incoming event.
  + *One-way Message Event* (also referred to as *Request Event*), which corresponds to a message sent pro-actively from the environment, e.g., a user request.
  + *Task Termination Event*, which corresponds to the message sent into an artifact instance upon termination of a previously invoked task.
- *Internal Event, a.k.a., Status Change Event:* These correspond to when a milestone changes status from false to true, or visa-versa, or when a stage changes status from inactive to active, or visa-versa.

Sentry: An expression in a condition language, that can refer to incoming events, internal events, and the values of data and status attributes. Sentries provide the core of the ECA-like rules that govern the progress of an artifact instance. In practical settings, a sentry will have the form **on** <*event*> **if** <*condition*> **then** <*action*>, where either the event or condition may be omitted. In the formalism here, a syntactic variant of this is used.

Milestone: A business-relevant operational objective that can be achieved by an artifact instance. In the artifact information model milestones are represented as named Boolean attributes. At a given moment in time a milestone has status *true* or *false*. Milestones have associated *achieving sentries* that determine situations under which the milestone becomes "achieved" and changes to status *true*, and *invalidating sentries* that determine situations under which the milestone becomes "invalidated" and change to status *false*.

Stage: Intuitively, a stage is a cluster of business-relevant activity that might be performed in connection with an artifact instance. Stages are organized into a hierarchy, to provide a rich form of modularity for specifying the overall behavior of an artifact instance. Each atomic stage contains exactly one task. Each stage "owns" one or more milestones, and the intuitive goal of executing a stage is to achieve one of these milestones. At a given moment in time a stage may have status *active* (or "open"), which corresponds to when the stage is executing, or *inactive* (or "closed"), which corresponds to when the stage is not executing. In the meta-model of the current paper, a stage may execute multiple times in sequence, but cannot have two occurrences that are executing simultaneously. (However, different stages might be executing simultaneously.)

***Guard*:** Guards are used to control whether a stage should become active, i.e., whether it should begin executing. Each guard is specified as a sentry. Guards are un-named, and their status cannot be referred to.[4]

Lifecycle model: The lifecycle model is a component that specifies the milestones and stages of a GSM artifact model, including their relationships (stage hierarchy, association of milestones to stages, and association of tasks to atomic stages), and the sentries that govern the guards and milestones.

2.2. Key concepts of GSM operational semantics

Here we introduce the key concepts relevant for the GSM operational semantics which will be in detail described in Sections 4 and 5.

Processing of incoming events: We assume in the GSM operational semantics that incoming events are processed one at a time. If more events occur at the same time we assume that it is the role of the implementing system to take care of managing the queue of incoming events.

GSM invariants: In the GSM meta-model of this paper, stages and milestones satisfy two properties that correspond to central business-level intuitions concerning how they interact. These are:

GSM-1 If a stage S owns a milestone m, then it cannot happen that both S is active and m has status *true*. In particular, if S becomes active then m must change status to *false*, and if m changes status to *true* then S must become inactive.

GSM-2 If stage S becomes inactive, the executions of all substages of S also become inactive.

Snapshot: A snapshot is an instantaneous description of an artifact instance at some moment in time. The snapshot includes the values of all data attributes and also the status of all milestones and stages of that time. In fact, an artifact instance can be thought of as the sequence of snapshots that it moves through during its lifetime.

Business step (B-step): B-step corresponds to the atomic unit of business-relevant processing in a GSM model. More specifically, it corresponds to incorporating into a GSM instance a single event incoming from the external environment, and then "firing" all of the sentries that might be applicable to the snapshot after the incoming event is incorporated.

2.3. An example GSM model

The GSM constructs are now illustrated through a series of examples.

[4]This is analogous to the fact that the individual achieving or invalidating sentries of milestones are not named. This design choice is based on experience in practical situations, where business process model designers see naming of the guards as unimportant.

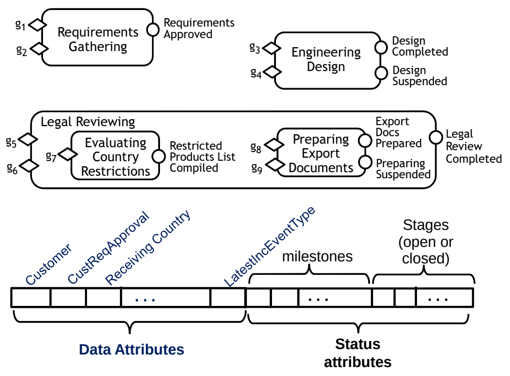


Figure 1: Graphical representation of parts of the Design-to-Order GSM model with its information model with data and status attributes, and its lifecycle model. Circles represent milestones, rounded rectangles represent stages, and diamonds represent guards.

Guard	Sentry
g_1	LatestIncEventType = Request : NewOrder
g_2	LatestIncEventType = Request : CustomerChange
g_3	RequirementsApproved ∧ RestrictedProductsListCompiled ∧ ¬DesignCompleted
g_4	LatestIncEventType = Request : ResumeEngineeringDesign
g_5	LatestIncEventType = Request : NewOrder
g_6	LatestIncEventType = Request : RedoExportDocuments
g_7	¬RestrictedProductsListCompiled
g_8	+DesignCompleted
g_9	LatestIncEventType = Request : RedoExportDocuments

Figure 2: Guards of Design-to-Order GSM model

RequirementsApproved *achieving*: LatestIncEventType = Termination : RequirementsGathering
DesignCompleted *achieving*: LatestIncEventType = Termination : EngineeringDesign *invalidating:* −RequirementsApproved
DesignSuspended *achieving*: −RequirementsApproved
RestrictedProductsListCompiled *achieving*: LatestIncEventType = Termination : EvaluatingCountryRestrictions
ExportDocsPrepared *achieving*: LatestIncEventType = Termination : PreparingExportDocuments *invalidating:* −DesignCompleted
PreparingSuspended *achieving*: −DesignCompleted
LegalReviewCompleted *achieving*: +ExportDocsPrepared *invalidating:* −ExportDocsPrepared

Figure 3: Sentries associated with milestones of Design-to-Order GSM model

Example 2.1. ***Order-to-Design Overview.*** Figures 1, 2, and 3 together illustrate an example GSM model, called *Design-to-Order*. Figure 1 provides a diagrammatic view; Figure 2 lists the sentries associated with the guards, and Figure 3 lists, for each milestone, its achieving and/or invalidating sentries. In this example, each milestone has at most one achieving sentry and at most one invalidating sentry. However, in the general case, multiple achieving and invalidating sentries are permitted.

The bottom part of Figure 1 shows the information model, which includes data attributes and status attributes. Status attributes are associated to the achievement status of a milestone (*true* or *false*) and to the activity state of stages (indicating whether they are active or inactive). The top half of the diagram shows the lifecycle specification. In the graphical notation milestones are shown as circles, stages are shown as rounded boxes, and guards are shown as diamonds.

This example models a design process that includes activities for requirements gathering, engineering design and legal review. We consider five actors in this process: customer, sales manager, request manager, engineering manager, and legal manager. The top-level activities, represented as stages in the GSM model, have the following functions.

Requirements gathering. This starts when a request with customer's requirements is received from the sales manager. It is considered completed when the requirements have been gathered and appropriate approvals obtained. The activity may be restarted if the customer wants to change the requirements.

Engineering design. This starts when the requirements are ready and the country restrictions have been evaluated as part of the legal review. It is considered completed when the design is created and appropriate approvals obtained. It might be suspended if the customer wants to change the requirements. The Engineering Design stage might need to be re-opened after it has successfully completed; this arises if the customer wants to change the requirements.

Legal review. The activity of this stage is divided between two sub-stages: Evaluating Country Restrictions, and Preparing Export Documents. The first starts as soon as the request for a new order is received. The second starts when the Engineering Design stage has completed successfully. Preparing Export Documents stage might have to be suspended or re-done if the customer requirements change.

▮

In GSM, a fundamental step in designing an artifact type is choosing the milestones that are typically achieved during the lifetime of instances of that type. The stages provide a natural mechanism for thinking in terms of groupings of work at various levels of detail. The stage hierarchy enables designers and business-level users to work with artifact lifecycles at varying levels of abstraction. In general, the milestones of higher-level stages correspond to the more visible, more business-important goals that an artifact instance might achieve. In practice for modestly sized examples the stage hierarchy is typically 3 to 5 levels deep. The running example is now used to illustrate the use of milestones and stages in GSM.

Example 2.2. ***Milestones and Stages.*** In Design-to-Order, the three most important milestones are 'Requirements Approved', 'Design Completed', and 'Legal Review Completed'. It is then natural to choose to have three top-level stages, one per major milestone. In the example, these stages are 'Requirements Gathering', 'Engineering Design', and 'Legal Reviewing'. Stage 'Engineering Design' has an additional milestone called 'Design Suspended'. 'Design Suspended' is included for cases where 'Requirements Approved' has been achieved and 'Engineering Design' launched, but then the customer presents new requirements. (See Example 2.4 for more detail on the full set of actions that might take place in this situation.)

In this simplified example, the stages 'Requirements Gathering' and 'Engineering Design' are atomic. This means that at the level of the GSM model, their internal functioning is a "black box", which is modeled as a task to be performed by an external agent (either human or automated). In the example in Figure 1 the tasks inside the atomic stages are omitted. Stage 'Legal Review' is a composite stage and it has two atomic sub-stages. This decomposition is motivated by the observation that there are two key milestones — 'Restricted Products List Compiled' and 'Export Docs Prepared' — that must be achieved before 'Legal Review Completed' can be achieved.

▮

While stages provide an intuitive form of encapsulation, they do not provide strict encapsulation in the way that, e.g., abstract data types, objects in object-oriented programming, or some web services paradigms do. Intuitively, the boundaries of stages are "soft", in the sense that sentries in any part of a lifecycle model may refer to the status of, and to attributes written by, any stage. For example, the guard g_3 of top-level stage 'Engineering Design' (see Figure 2) refers to the milestone 'Restricted Products List Compiled', which is inside the 'Legal Reviewing' stage. Changes in a status attribute in one part of the stage hierarchy may also trigger sentries elsewhere in the hierarchy (e.g., guard g_8). The ability of stages to "see" data and status change events from essentially anywhere in the stage hierarchy is one aspect of the data-centric nature of the artifact approach.

The next example illustrates the event types supported in GSM and the central construct called 'sentry'.

Example 2.3. ***Sentries and Event Types.*** Figures 2 and 3 show the sentries associated with the guards (Figure 2) and milestones (Figure 3) of Design-to-Order GSM model. Although guards are un-named in GSM, in the Figures 1 and 2 we have included names $g_1, g_2, \ldots$ for ease of exposition.

Intuitively, sentries have the form "**on** $\langle$ *event* $\rangle$ **if** $\langle$ *condition* $\rangle$," which corresponds to the first two parts of traditional event-condition-action rules. In GSM, the event may be an incoming event type or a status change event type, and the condition may refer to data and status attributes in the information model.

In the formal notation, incoming events are specified using the dedicated data attribute $\mathsf{LatestIncEventType}$, which is intended to hold the type of the most recently arrived incoming event. For example, the guard g_1 of 'Requirements Gathering' is specified as "$\mathsf{LatestIncEventType} = \mathsf{Requests} : \mathsf{NewOrder}$". This sentry becomes true if there is an incoming event with type 'Request: New Order'.

Analogously, the milestone ‘Requirements Approved’ is achieved when an event of type ‘Termination: Requirements Gathering’ is received, i.e., when an occurrence of the atomic task associated with ‘Requirements Gathering’ terminates.

We next illustrate a sentry triggered by a change in value of a status attribute. The guard g_8 of ‘Preparing Export Documents’ is specified as “$+\mathsf{DesignCompleted}$”; this is triggered when milestone ‘Design Completed’ is achieved, i.e., when it transitions from *false* to *true*. Similarly, an invalidating sentry of milestone ‘Design Completed’ is “$-\mathsf{RequirementsApproved}$”, that is, if the ‘Requirements Approved’ milestone of ‘Requirements Gathering’ is invalidated, then so is the ‘Design Completed’ milestone.

Some sentries do not have a triggering event. Guard g_3 of ‘Engineering Design’ is specified as “$\mathsf{RequirementsApproved} \wedge \mathsf{RestrictedProductsListCompiled} \wedge \neg\mathsf{DesignCompleted}$”; this is based on having two milestones be true and a third one false. Permitting sentries such as this, which do not explicitly refer to a change in status attribute, can enable more flexibility and succinctness. Because guard g_3 does not involve a triggering event, it is important to prevent the ’Engineering Design’ stage from unintended openings after it has successfully completed. This is accomplished in this sentry by including the third conjunct $\neg\mathsf{DesignCompleted}$. (An alternative would be to adopt a semantics based on “when first becomes true” for condition-based guards. In this semantics, such guards would trigger when their condition shifts from false to true. Study of this variation is left for future research.) ▮

The final example illustrates the notion of B-steps. Recall that intuitively, a B-step corresponds to incorporating one incoming event into an artifact instance, and then “firing” sentries to change the status of milestones and/or stages until no further sentries can be applied.

Example 2.4. ***Illustration of B-steps.*** This example describes three representative B-steps. All of them start with events from the customer. The first example describes what happens when a new order arrives, the second describes what happens when an ‘Engineering Design’ stage gets suspended, and the final one describes what may happen if the customer wants to incorporate additional requirements.

Initial steps in Design-to-Order artifact instance. Upon receiving an event of the type $\mathsf{Request:NewOrder}$ the guard g_1 of ‘Requirements Gathering’ opens that stage and invokes the associated task. In the same B-step, the composite stage ‘Legal Review’ is opened by the guard g_5 with condition “$\mathsf{LatestIncEventType} = \mathsf{Request:NewOrder}$”. This makes all stages inside ‘Legal Review’ eligible. In particular, the guard g_7 of stage ‘Evaluating Country Restriction’ with condition $\neg\mathsf{RestrictedProductListCompiled}$ opens its stage since the mentioned milestone is false, and this in turn invokes the task inside the stage. Thus, the B-step resulting from the incoming event involves opening three stages and launching of two tasks.

Suspending ‘Engineering Design’. First, suppose that ‘Requirements Approved’ is true and that ‘Engineering Design’ is active. Suppose further that the customer now submits a change in requirements, i.e., an event of form $\mathsf{Request:CustomerChange}$ arrives. This triggers guard g_2 and leads to re-opening the stage ‘Requirements Gathering’. This in turn leads to invalidation of milestones ‘Requirements Approved’, based on the policy that you cannot simultaneously have a stage active and one of its milestones true. Next, the ‘Design Suspended’ milestone is achieved because its sentry is triggered by invalidation event of ‘Requirements Approved’ milestone. This in turn leads to closing the stage ‘Engineering Design’ (i.e., its execution is halted), again based on the policy that you cannot simultaneously have a stage active and one of its milestones true.[5]

Re-doing legal work. As a final illustration, suppose that all three top-level stages have completed successfully, and then the customer submits new requirements. Over time this will lead to re-doing ‘Requirements Gathering’ and ‘Engineering Design’. When ‘Engineering Design’ starts it will automatically invalidate ’Design Completed’, which will in turn invalidate ‘Export Docs Prepared’ and then ‘Legal Review Completed’. When ‘Design Completed’ is achieved again the stage ‘Preparing Export Documents’ will not be re-opened, because its parent stage ‘Legal Reviewing’ is currently inactive. In the Design-to-Order GSM model, the legal manager can issue a request of form $\mathsf{Request} : \mathsf{RedoExportDocuments}$. This will trigger g_6 to open ‘Legal Reviewing’ and also g_9 to open ‘Preparing Export Documents’. ▮

As just illustrated, the propagation of actions in a B-step can become rather intricate. One intention of GSM is to enable business-level workers to specify large portions of a GSM model in terms of milestones, stages, and their associated sentries. Specialized business analysts may be needed to design the more intricate interactions between the sentries, but in most cases the chain of actions in response to an incoming event can still be explained to the business-level workers. A primary contribution of the current paper is to provide a robust, formal foundation so that for any well-formed GSM model there is an unambiguous and intuitively natural semantics that underlies the response to incoming events.

3. Formal basis for GSM

This section introduces the formal definitions for most of GSM, leading to the central notion of GSM model.

3.1. Names, attributes, domain, condition language

We assume infinite, pairwise disjoint sets $\textsc{Event}$ of (*incoming*) *event names*, $\textsc{Stage}$ of *stage names*, and $\textsc{Milestone}$ of *milestone names*.

In order to represent artifact information models we assume disjoint sets $\textsc{Att}_{\mathit{data}}$ of *data attribute names*, or simply, *data attributes*, and $\textsc{Att}_{\mathit{status}}$ of *status attribute names*, or simply, *status attributes*. Data attributes are used to hold information about the business operations that are being managed by an artifact instance. The set $\textsc{Att}_{\mathit{status}}$ = $\textsc{Milestone} \cup \{\mathit{active}_S \mid S \in \textsc{Stage}\}$ (where we assume $\textsc{Milestone} \cap \{\mathit{active}_S \mid S \in \textsc{Stage}\} = \emptyset$). Status attributes help to record the current status of an artifact

[5]In practice, the task owned by ‘Engineering Design’ might itself be suspended rather than aborted, so that work already performed as part of the task is not lost; this is not addressed in the abstract GSM model presented in this paper.

instance. A milestone name m, when considered as an attribute, will take Boolean values, and will indicate whether the milestone m is currently true or not. An attribute $\textit{active}_S$ for $S \in \textsc{Stage}$ will take Boolean values, and will indicate whether the stage S is active or inactive.

For ease of presentation, we consider attributes to have a single common domain $\textsc{Dom}$, which includes an *undefined* symbol $\perp$, a set of elements that represent numbers and strings, and the two Boolean constants $\{\textit{true}, \textit{false}\}$. We assume further that $\textsc{Dom}$ includes $\textsc{Event}$. (Intuitively, this will allow for expressions of the form $\textit{LatestIncEventType} = E$ where E is an event type and *LatestIncEventType* is an attribute intended to hold the type of the incoming event that arrived most recently to an artifact instance.)

If $\mathcal{A}$ is a finite set of data attributes, a *logical structure* over $\mathcal{A}$ is an assignment $\Xi : \mathcal{A} \rightarrow \textsc{Dom}$. (In what follows, such structures correspond intuitively to snapshots of artifact instances.) For this paper we shall use a condition language $\mathcal{C}$ that refers to attributes in *Att* as variables (where $\textit{Att} \subset \textsc{Att}_{\textit{data}} \cup \textsc{Att}_{\textit{status}}$). For instance, if $\mathsf{completed}$ and price are attributes in *Att*, then $\mathsf{completed} = true \wedge \mathsf{price} = \$50$ is a condition. In the rest of the paper, when it is clear from the context that an attribute a holds a Boolean value we use the notation a and $\neg\mathsf{a}$ to mean $\mathsf{a} = \textit{true}$ and $\mathsf{a} = \textit{false}$, respectively. For the purposes of the present paper we assume that $\mathcal{C}$ is a propositional language, with atoms of the form $x\theta y$ where $x, y \in \mathcal{A} \cup \textsc{Dom}$ and θ is a binary relation symbol. (The particular set of relation symbols supported does not matter, as long as = is present.)

Remark 3.1. We note that the results of this paper remain true if the artifact instances are considered within a larger context, e.g., one that includes a family of database relations in the spirit of [15]. In such cases, the condition language $\mathcal{C}$ used may include variables other than attributes in $\mathcal{A}$, and may include quantification over those variables. ▮

In this paper it is convenient to work with pairs (Ξ, Ξ') of structures, where intuitively Ξ' is constructed at some point in time after Ξ was constructed. In such cases, following convention from the verification community, we may use formulas that refer to attributes in $\mathcal{A}$ and to *primed attributes*, i.e., expressions of the form A' where $A \in \mathcal{A}$, that are used to refer to the values associated to attributes in Ξ'. Given a formula $\varphi(x_1, \ldots, x_n, y'_1, \ldots, y'_m)$ where $x_1, \ldots, x_n, y_1, \ldots, y_m \in \mathcal{A}$, the pair (Ξ, Ξ') satisfies φ, denoted $(\Xi, \Xi') \models \varphi$, if $\varphi[x_1/\Xi(x_1), \ldots, x_n/\Xi(x_n), y'_1/\Xi'(y_1), \ldots, y_m/\Xi'(y_m)]$ evaluates to true.

3.2. Tasks, Messages, Events

We now turn to the kinds of events that can pass between an artifact instance and the environment. These event types are in two categories

- *Incoming Event Types*: these include types for *one-way messages* and *task terminations*
- *Outgoing Event Types*: these include types for *task invocations*

To formalize these we first consider incoming one-way messages, and then tasks.

Intuitively, incoming one-way messages are unsolicited events from the environment that are used to model direct user requests (e.g., request to create a new order, request to revise requirements, manager approval) and other spontaneous phenomena occurring in the external environment.

We assume a set $\textsc{Message}$ of *message names*, that is disjoint from the other sets of names already established.

Definition 3.2. An incoming *one-way message* (*event*) *type* is a triple $E = \langle M, O, \psi \rangle$, where

- $M \in \textsc{Message}$ is a message name,
- $O \subseteq \textsc{Att}_{\mathit{data}}$ is a message payload structure, and
- ψ is a condition that refers to attributes in O.

It is assumed that two distinct message types have distinct names, i.e., if $\langle M, O, \psi \rangle$ and $\langle M,\, O', \psi' \rangle$ are message types, then $O = O'$, and ψ and ψ' are identical. We sometimes refer to message type $\langle M, O, \psi \rangle$ simply as M. The set of one-way message event types is denoted $\textsc{Event}_{\mathit{msg}}$.

A *one-way message event* of one-way message type $E = \langle M, O, \psi \rangle$ is a pair $e = \langle M, p \rangle$ where $p : O \to \textsc{Dom}$ is the *payload* and $\psi[p]$ evaluates to true. ∎

The condition ψ in a one-way message type corresponds intuitively to restrictions that are known to apply to payloads of messages of that type.

Tasks model both computer-executed tasks as well as human-performed ones. Following the spirit of semantic web services [32] and also the earlier work on declarative artifact meta-models [7, 15, 12], tasks include input and output attributes, and postconditions. (We do not include preconditions, because the guards on atomic stages holding tasks will take that role.)

We assume a set $\textsc{Task}$ of *task names*, that is disjoint from the other sets of names already established.

Definition 3.3. A *task* is a tuple $\langle T, I, O, \psi \rangle$, where

- $T \in \textsc{Task}$ is a task name,
- $I \subseteq \textsc{Att}_{\mathit{data}}$ (*input attributes*),
- $O \subseteq \textsc{Att}_{\mathit{data}}$ (*output attributes*), and
- ψ (*postcondition*) is a logical formula in $\mathcal{C}$ referring to attributes in I without primes and to attributes in O with primes.

It is assumed that two distinct tasks have distinct names, i.e., if $\langle T, I, O, \psi \rangle$ and $\langle T, I', O', \psi' \rangle$ are tasks, then $I = I'$, $O = O'$, and ψ and ψ' are identical. We sometimes refer to task $\langle T, I, O, \psi \rangle$ simply as T.

A *task invocation event type* is a pair $E = \langle T, I \rangle$ where $\langle T, I, O, \psi \rangle$ is a task. A *task invocation event* of this type is a pair $e = \langle T, p \rangle$ where $p : I \rightarrow \text{DOM}$ is the *input payload*. The set of task invocation event types is denoted EVENT_{inv}.

A *task termination event type* is a triple $E = \langle T, I, O \rangle$ where $\langle T, I, O, \psi \rangle$ is a task. A *task termination event* of this type is a triple $e = \langle T, p, p' \rangle$ where $\langle T, p \rangle$ is a task invocation event, $p' : O \rightarrow \text{DOM}$, and $\psi[p, p']$ evaluates to true. Here p is called the *input payload* of e and p' is called the *output payload* of e. The set of task termination event types is denoted EVENT_{term}. ▮

Intuitively, task invocation events occur when an artifact instance invokes a task, and task termination events occur when the invoked task is completed. The assignment p will correspond to the values of attributes in I of the invoking artifact instance, and assignment p' will cause the assignment of values to attributes in O of the artifact instance at the time when the task terminates. Postconditions model constraints on the outcomes of task executions. They can model both (i) deterministic tasks (e.g. performing a computation: $x' = x+1$), and (ii) black-box services or human performed tasks (e.g. stating that an attribute is only guaranteed to be filled: $x' \neq \bot$; or that an attribute lies within some range: $20 \leq y' \wedge y' \leq 30$).

It is convenient to refer to the full set of incoming types.

Definition 3.4. The set of *incoming event types* is $\text{EVENT}_{inc} = \text{EVENT}_{term} \cup \text{EVENT}_{msg}$. It is assumed that $\text{EVENT}_{inc} \subseteq \text{EVENT}$. ▮

3.3. GSM data model and pre-snapshot

We introduce the notion of GSM model in two steps. The first is focused on the data structure associated with a GSM model. This structure will allow us to define the important notion of 'sentry', which form the core of the ECA-like rules that are used in GSM lifecycle models.

Definition 3.5. A *GSM data model* is a triple $\Delta = (\mathit{Att}, \mathcal{S}, M)$ where the following hold.

1. *Att* $\subset \text{ATT}_{data} \cup \text{ATT}_{status}$ is finite, called the *attributes* of Δ. (Recall that $\text{ATT}_{data} \cap \text{ATT}_{status} = \emptyset$.)
2. $\mathcal{S} \subset \text{STAGE}$ is finite, called the *stages* of Δ.
3. $M \subset \text{MILESTONE}$ is finite, called the set of *milestones* of Δ.
4. *Att* includes a data attribute called *LatestIncEventType.*
5. $M \subset$ *Att*, i.e., each milestone name $m \in M$ also occurs in *Att* as a status attribute.
6. $\{\mathit{active}_S \mid S \in \mathcal{S}\} \subset$ *Att*, i.e., for each stage name $S \in \mathcal{S}$, the status attribute active_S occurs in *Att*.

▮

Intuitively, a GSM data model $\Delta = (\mathit{Att}, \mathcal{S}, M)$ provides the skeleton for GSM artifact instances, including the basic control structure provided by stages and milestones. In particular, the data attributes in *Att* are used to hold business-relevant information about the progress of the artifact instance through the business operations. The milestone attributes $m \in M$ will take Boolean values, and will record, at a given moment in time, whether m has most recently been achieved or invalidated. And attributes active_S for $S \in \mathcal{S}$ will take Boolean values, and will record whether stage S is currently active or inactive. As described in Subsection 3.5 below, a GSM lifecycle model associated with a GSM data model will specify relationships between the stages and milestones.

We now introduce the notion of 'pre-snapshot' for a GSM data model; this corresponds to an instantaneous description of a running instance of the GSM data model at a given point in time. (This notion is extended to GSM models in Definition 3.12.)

Definition 3.6. A *pre-snapshot* of a GSM data model $\Delta = (\mathit{Att}, \mathcal{S}, M)$ is an assignment function $\Sigma : \mathit{Att} \rightarrow \textsc{Dom}$ where

(i) $\Sigma(a) \in \{\mathit{true}, \mathit{false}\}$ for each status attribute $a \in \mathit{Att}$, and
(ii) $\Sigma(\mathit{LatestIncEventType}) \in \textsc{Event}_{\mathit{inc}}$.

▮

3.4. Status event and Sentry

In addition to incoming events, it is useful to model events corresponding to changes in the status attributes of an artifact instance.

Definition 3.7. Let $\Delta = (\mathit{Att}, M, \mathcal{S})$ be a GSM data model. A *status change event*, or simply *status event*, for Δ is an expression of form $+a$ or $-a$ where a is a status attribute for Δ, i.e., $a \in \mathit{Att} \cap \textsc{Att}_{\mathit{status}}$. The set of status events for Δ is denoted as $\mathcal{E}_{\mathit{status}}(\Delta)$ ▮

We typically use the symbol '$\odot$' to denote the *polarity* of a status event, i.e., $\odot \in \{+, -\}$.

Recall that in the condition language $\mathcal{C}$ used in this paper, data attributes are used as variables. We shall also use status events as Boolean variables in expressions in $\mathcal{C}$. The intuitive meaning of $+a$ is that a has shifted from false to true during the course of a B-step, and analogously for $-a$.

Definition 3.8. Let $\Delta = (\mathit{Att}, \mathcal{S}, M)$ be a GSM data model, and Σ, Σ' be pre-snapshots of Δ. If a is a status attribute in *Att* then (Σ, Σ') *satisfies* $+a$, written $(\Sigma, \Sigma') \models +a$ if $\Sigma \models \neg a$ and $\Sigma' \models a$. Similarly, (Σ, Σ') *satisfies* $-a$, written $(\Sigma, \Sigma') \models -a$ if $\Sigma \models a$ and $\Sigma' \models \neg a$. The meaning of $(\Sigma, \Sigma') \models \varphi$, where φ is a formula in $\mathcal{C}$ possibly involving status events and both unprimed and primed attribute symbols, is defined in the natural manner. ▮

Intuitively, a status event $+a$ can be viewed as a macro for the condition $\neg a \wedge a'$, and the status event $-a$ can be viewed as a macro for the condition $a \wedge \neg a'$.

We can now define the notion of 'sentry', which forms the core of the ECA-like rules used in GSM.

Definition 3.9. Let $\Delta = (\mathit{Att}, \mathcal{S}, M)$. A *sentry* for Δ is a Boolean formula of the form $\tau \wedge \gamma$, where

1. γ is a formula that contains no incoming event types or status events, and
2. τ is (a) empty, or (b) has the form $\mathsf{LatestIncEventType} = E$ for some incoming event type E, or (c) has the form $\odot a$ for some status attribute a and $\odot \in \{+, -\}$. In case (b), E is the *trigger* of this sentry, and in case (c), $\odot a$ is the *trigger* of this sentry.

The set of sentries for Δ is denoted $\textsc{Sentry}(\Delta)$. ▮

If a sentry is written in form $\tau \wedge \gamma$ as above, then it is easy to transform it into the classical form "**on** $\langle$ *event* $\rangle$ **then** $\langle$ *condition* $\rangle$."

Remark 3.10. Recall Remark 3.1, which stated that the results of this paper remain true if artifact instances are considered within a larger context, e.g., one that includes a family of database relations. In that case, the notion of 'sentry' can be expanded by permitting the formulas γ in Definition 3.9 be closed formulas including quantified variables ranging over elements of the larger context. ▮

3.5. GSM model, snapshot, and environment

A GSM model includes a GSM data model and a lifecycle model.

Definition 3.11. A *GSM model* is a tuple $\Gamma = (\mathit{Att}, \mathcal{S}, M, \mathcal{L})$, where $(\mathit{Att}, \mathcal{S}, M)$ is a GSM data model, and where the *lifecycle model* $\mathcal{L}$ of Γ has structure $(\mathit{Substages}, \mathit{Task}, \mathit{Owns}, \mathit{Guards}, \mathit{Ach}, \mathit{Inv})$ and satisfies the following properties.

1. $\mathit{Substages} : \mathcal{S} \to \mathcal{P}(\mathcal{S})$ is a function from $\mathcal{S}$ to finite subsets of $\mathcal{S}$, such that the relation $\{(S, S') \mid S' \in \mathit{Substages}(S)\}$ creates a forest. A root of this forest is called a *top-level stage*, a leaf is called an *atomic stage*, and a non-leaf node is called a *composite stage*. The set of atomic stages is denoted $\mathcal{S}_{\mathit{atomic}}$.
2. $\mathit{Task} : \mathcal{S}_{\mathit{atomic}} \to \textsc{Task}$ is an injection.
3. $\mathit{Owns} : \mathcal{S} \to \mathcal{P}^{\mathit{nonempty}}(M)$ (i.e., mapping to nonempty subsets of M) such that $\mathit{Owns}(S) \cap \mathit{Owns}(S') = \emptyset$ for $S \neq S'$. A stage S *owns* a milestone m if $m \in \mathit{Owns}(S)$.
4. $\mathit{Guards} : \mathcal{S} \to \mathcal{P}^{\mathit{fin}}(\textsc{Sentry}(\mathit{Att}, M, \mathcal{S}))$ (i.e., mapping to finite subsets of sentries over the data model of Γ).
5. $\mathit{Ach} : M \to \mathcal{P}^{\mathit{fin}}(\textsc{Sentry}(\mathit{Att}, M, \mathcal{S}))$. For milestone m, each element of $\mathit{Ach}(m)$ is called an *achieving sentry* of m.
6. $\mathit{Inv} : M \to \mathcal{P}^{\mathit{fin}}(\textsc{Sentry}(\mathit{Att}, M, \mathcal{S}))$. For milestone m, each element of $\mathit{Inv}(m)$ is called an *invalidating sentry* of m.

▮

We now extend the notion of pre-snapshot for GSM data models to full GSM models. We also define the notion of 'snapshot', which is a pre-snapshot satisfying some internal consistency conditions. These correspond to the intuitions that (a) once a milestone is achieved then the owning stage should be terminated, and (b) if a stage is terminated all of its children should be terminated.

Definition 3.12. Let $\Gamma = (\textit{Att}, \mathcal{S}, M, \mathcal{L})$ be a GSM model. Each pre-snapshot of $(\textit{Att}, \mathcal{S}, M)$ is also said to be a *pre-snapshot* of Γ. A pre-snapshot of Γ is a *snapshot* of Γ if it satisfies the following *GSM invariants*:

- *GSM-1: Stage and milestone cannot both be true.* If $m \in \textit{Owns}(S)$ and $\Sigma(\textit{active}_S) = \textit{true}$, then $\Sigma(m) = \textit{false}$
- *GSM-2: No activity in closed stage.* If $\Sigma(\textit{active}_S) = \textit{false}$ for stage $S \in \mathcal{S}$ and $S' \in \textit{Substages}(S)$, then $\Sigma(\textit{active}_{S'}) = \textit{false}$.

▮

Remark 3.13. In some variations of the GSM meta-model [24, 23] a third invariant *GSM-3* is assumed, which states that for each stage S, at most one of its milestones can be true. This invariant is enforced by some form of syntactical restriction on the milestone achieving conditions. To streamline presentation here, and without loss of generality, we do not consider *GSM-3* here.

▮

We conclude the subsection by formalizing a notion of 'environment' that is used to model the part of the real-world environment relevant to the operation of a GSM artifact instance. In particular, it is assumed that the environment will perform the tasks that are invoked from the artifact instance, and send termination events back to the instance when a task completes. In the current GSM meta-model, each stage can have at most one active occurrence at a given time (although it may have multiple occurrences that occur sequentially through time). Thus, to model the environment of a GSM model $\Gamma = (\textit{Att}, \mathcal{S}, M, \mathcal{L})$, it is sufficient to incorporate, for each task T occurring in the range of *Task*, variables to hold the values of the input attributes used in an invocation of task T.

As a technical convenience we first extend the set of attributes as follows.

Definition 3.14. An *environment attribute* is a symbol x_T where $\langle T, I, O, \psi \rangle$ is a task and $x \in I$. The set of environment attributes is denoted $\text{ATT}_{\textit{env}}$. ▮

Definition 3.15. The *environment* for GSM model $\Gamma = (\textit{Att}, \mathcal{S}, M, \mathcal{L})$ is the set of attributes $\text{ENV}_\Gamma = \{x_T \mid T = \textit{Task}(S), S \in \mathcal{S}_{\textit{atomic}}\}$. A *value* for this environment is a function $\Omega : \text{ENV}_\Gamma \rightarrow \text{DOM}$. ▮

4. PAC rules and Well-formed GSM Models

This section presents the building blocks used to define the operational semantics of GSM models. The section starts with an informal description of the notion of GSM Business Step (B-step). It then describes Precedent-Antecedent-Consequent (PAC) rules, a variation of ECA rules used to capture the intended meaning of the sentries used in a GSM model, and also the GSM invariants. Using examples (Examples 4.6 and 4.7), we illustrate potential non-intuitive behaviors that can be exhibited by unrestricted application of PAC rules. Finally, the section introduces the Polarized

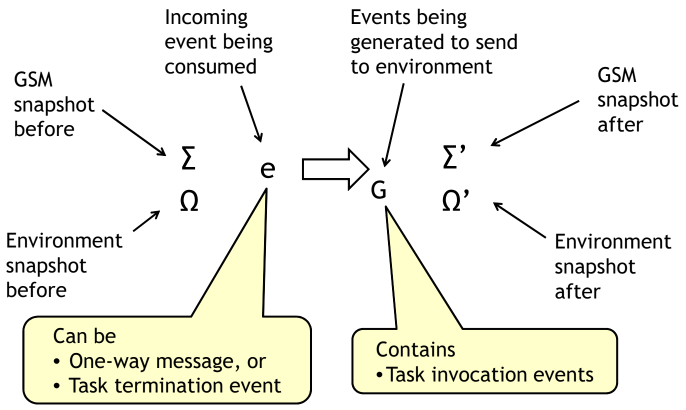

Incoming event being consumed
Events being generated to send to environment
GSM snapshot before
GSM snapshot after
Σ
Ω
e
G
Σ'
Ω'
Environment snapshot before
Environment snapshot after
Can be
• One-way message, or
• Task termination event
Contains
•Task invocation events

Figure 4: Illustration of a single GSM Business step (B-step)

Dependency Graph (PDG) of a GSM model. The PDG is used to define the well-formedness condition for GSM models (based on an acyclicity of the PDG), and to structure the application of the PAC rules, with the goal of avoiding the non-intuitive behaviors and thus providing unambiguous and intuitively natural semantics that underlies the response to incoming events.

4.1. Informal Description of GSM Business steps (B-steps)

The operational semantics for GSM are focused on the notion of B-steps, which capture the impact of a single incoming event occurrence e on a snapshot Σ of a GSM model Γ. In Section 5 three formal notions of B-step are defined and shown to be equivalent. The term 'B-step' will then be defined to include the B-steps of each (any) of the three formal notions.

The notion of B-step is illustrated in Figure 4. The semantics characterizes 6-tuples of the form $(\Sigma, e, \Sigma', \Omega, G, \Omega')$, where the following hold.

1. Σ is the *previous* snapshot.
2. e is a ground occurrence of an incoming event type associated with Γ.
3. Σ' is the *next* snapshot.
4. Ω is the *previous* value of the environment.
5. G is the set of ground *generated event occurrences*, all of whom have task invocation type. (These arise when atomic stages become active.)
6. Ω' is the *next* value of the environment.

To illustrate the notion of B-step, we describe key aspects of the *incremental formulation* of the operational semantics. (This is formalized in Definition 5.1 below.) In this case, Σ' is constructed in two phases (see Figure 5). The first is to incorporate event e into Σ, by computing the "immediate effect" Σ^e of e on Σ. Intuitively, this has the effect of (a) updating the attribute *LatestIncEventType* to hold the type of e, and (b) updating the values of all data attributes directly affected by the payload of e.

We pause to give the formal definition of 'immediate effect.' To that end, we first introduce the notion of when an event is "applicable" to a GSM snapshot.

Definition 4.1. Let $\Gamma = (\mathit{Att}, \mathcal{S}, M, \mathcal{L})$ be a GSM model and Σ a snapshot of Γ. Incoming event e is *applicable* to Σ if either

1. $e = \langle M, p \rangle$ has type $E = \langle M, O, \psi \rangle \in \textsc{Event}_{\mathit{msg}}$ and $p \models \psi$, or
2. $e = \langle T, p, p' \rangle$ has type $E = \langle T, I, O, \psi \rangle \in \textsc{Event}_{\mathit{term}}$, $T = \mathit{Task}(S)$ for some atomic stage $S \in \mathit{Stages}$, $\Sigma(\mathit{active}_S) = \mathit{true}$, and $(p, p') \models \psi$.

∎

For both kinds of events, in order to be applicable the formula ψ must be satisfied by the associated assignments. Further, with regards to termination events $e = \langle T, p, p' \rangle$, the atomic stage S that owns task T must be active in snapshot Σ. (We note that S might become active, launch an occurrence of T, and then become inactive because some milestone of S unrelated to T goes true. In that case the occurrence of T should be abandoned, and any termination event from that occurrence should be ignored.)

Definition 4.2. Let $\Gamma = (\mathit{Att}, \mathcal{S}, M, \mathcal{L})$ be a GSM model and Σ a snapshot of Γ, and let $e \in \textsc{Event}_{\mathit{inc}}$ be applicable to Σ. The *immediate effect* of e on Σ, denoted Σ^e is defined as follows.

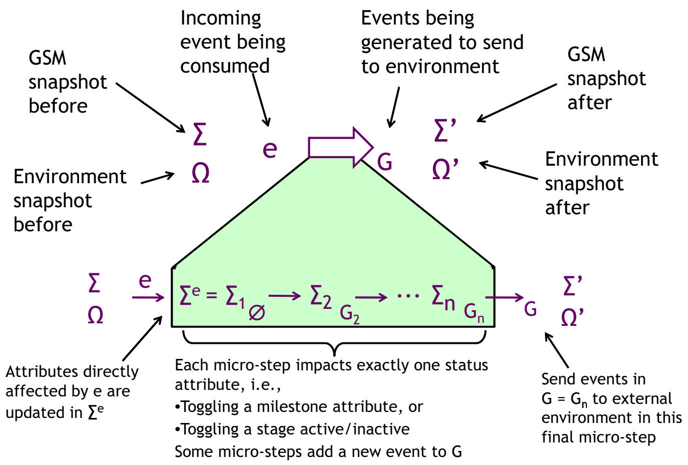
1. if $e = \langle M, p \rangle$ has type $E = \langle M, O, \psi \rangle \in \textsc{Event}_{\mathit{msg}}$, then Σ^e is defined so that $\Sigma^e(A) = p(A)$ for each attribute $A \in I$ and $\Sigma^e(A) = \Sigma(A)$ for all other attributes A.
2. if $e = \langle T, p, p' \rangle$ has type $E = \langle T, I, O, \psi \rangle \in \textsc{Event}_{\mathit{term}}$, then Σ^e is defined so that $\Sigma^e(A) = p'(A)$ for each attribute $A \in O$ and $\Sigma^e(A) = \Sigma(A)$ for all other attributes A.

∎

We now return to incremental construction of B-steps. The second phase is to incorporate the effects of the guards, achieving and invalidating sentries for milestones, and the two GSM invariants. A family of ECA-like rules corresponding to these constructs is derived from Γ (Subsection 4.2). A sequence $\Sigma = \Sigma_0, \Sigma^e = \Sigma_1, \Sigma_2, \ldots, \Sigma_n = \Sigma'$ of pre-snapshots is constructed, where (1) each step in the computation after Σ^e is called a *micro-step* and it corresponds to the application of one ECA-like rule, and where (2) no ECA-like rule can be applied to Σ_n. (The intermediate values $\Sigma_1, \ldots, \Sigma_{n-1}$ might violate GSM-1 or GSM-2, which is why they are permitted to be pre-snapshots as opposed to snapshots.) There are restrictions on the ordering of rule application, as detailed in Subsection 5.1 below. Finally, Σ' is returned as the result of the B-step. For each micro-step one also maintains a set G_j of *generated events*, which are sent to the environment at the termination of the B-step.

In order to streamline the presentation, in the rest of the paper we do not consider the generation of outgoing events, and assume that the environment is correctly updated at the end of each B-step. In particular, when discussing B-steps we shall focus on triples of the form (Σ, e, Σ') where Σ, Σ' are snapshots of the GSM model and e is an incoming event applicable to Σ.

Although the creation of Σ' from Σ and e may take a non-empty interval of clock time, in the formal model this is considered instantaneous. (In other, more practically oriented investigations of GSM [22, 23] the logical timestamp of each B-step is also maintained; this detail is omitted here because the focus is on properties of a single B-step.)



GSM snapshot before
Incoming event being consumed
Events being generated to send to environment
GSM snapshot after
Σ
Ω
e
G
Σ'
Ω'
Environment snapshot before
Environment snapshot after
Σ
Ω
e
Σ^e = Σ_1 ∅ → Σ_2 G_2 → ··· Σ_n G_n → G
Σ'
Ω'
Attributes directly affected by e are updated in Σ^e
Each micro-step impacts exactly one status attribute, i.e.,
•Toggling a milestone attribute, or
•Toggling a stage active/inactive
Some micro-steps add a new event to G
Send events in G = G_n to external environment in this final micro-step

Figure 5: Incremental formulation of GSM semantics

Intuitively, B-steps are considered to be natural “units of business-relevant change”. In naturally arising cases, a B-step will capture all of the effects of incorporating a single incoming event, including changes to milestone and stage status. While the construction of a snapshot Σ' from snapshot Σ and event e will be treated as a “black box” from the perspective of the business, the values of Σ and Σ' are considered business relevant. This perspective on B-steps will lead to some requirements on how they are defined (see Subsection 4.3).

4.2. Prerequisite-Antecedent-Consequent Rules

We now turn to the ECA-like rules associated with a GSM model. These are used to guide the micro-steps in the incremental computation of a B-step, and are also used in the other formulations of the GSM operational semantics.

Definition 4.3. A *precedent-antecedent-consequent* (or *PAC*) rule is a triple $\langle \pi, \alpha, \gamma \rangle$, where:

- π (*prerequisite*), is a formula in $\mathcal{C}$ on attributes in *Att*,
- α (*antecedent*), is a formula in $\mathcal{C}$ on attributes in *Att*, and
- γ (*consequent*), is a *status change event* of form $\odot a$, where $a \in Att_{status}$ and $\odot \in \{+, -\}$

∎

Figure 6 shows templates for the six kinds of PAC rules associated with a GSM model Γ. The first three templates correspond directly to sentries in Γ, and address opening stages, achieving milestones, and invalidating milestones. The last three templates correspond to maintaining the GSM invariants. PAC-5 and PAC-6 are straightforward. The primary role of PAC-4 is to ensure that when a stage opens then any true

	Basis	Prerequisite	Antecedent	Consequent
Explicit rules				
PAC-1	Guard: for each stage S, for each guard φ of S. (Include term *active*$_{S'}$ if S' is parent of S.)	$\neg$*active*$_S$	$\varphi \wedge$ *active*$_{S'}$	$+$*active*$_S$
PAC-2	Milestone achiever: For each milestone m of stage S with achieving sentry φ.	*active*$_S$	φ	$+m$
PAC-3	Milestone invalidator: For each milestone m of stage S with invalidating sentry φ.	m	φ	$-m$
Invariant preserving rules				
PAC-4	Guard invalidating milestone: For each guard φ of a stage S, for each milestone m not occuring in a top-level conjunct $\neg m$. (Include term *active*$_{S'}$ if S' is parent of S.)	m	$\varphi \wedge$ *active*$_{S'}$	$-m$
PAC-5	For each milestone m of a stage S.	*active*$_S$	$+m$	$-$*active*$_S$
PAC-6	For each stage S child of S'.	*active*$_S$	$-$*active*$_{S'}$	$-$*active*$_S$

Figure 6: Templates for PAC rules associated with a GSM model Γ. (Here '$\neg$' is conventional logical negation, '+' is used to indicate that a status attribute has/will transition from false to true, and '-' is used to indicate that a status attribute has/will transition from true to false.)

milestone owned by that stage is invalidated; this is half of enforcing invariant GSM-1. Remark 4.12 below motivates the particular formulation of PAC-4 used here.

In all but PAC-2, the prerequisite is a kind of "opposite" of the consequent. Intuitively, PAC-2 is different because of the very close relationship between stages and their milestones; see Remark 5.5 below.

Definition 4.4. Let Γ be a GSM model. The set of *PAC rules* for Γ, denoted rules_Γ, is the set of all PAC rules that are formed for Γ using the templates PAC-1, ..., PAC-6 shown in Figure 6. ▮

The following definition describes how PAC rules can be applied to pre-snapshots.

Definition 4.5. Let $\langle \pi, \alpha, \odot a \rangle \in \mathsf{rules}_\Gamma$ be a PAC rule. The *primed version* of α, denoted α', is formed from α by replacing each attribute A by A'. (The status events $\widehat{\odot}\widehat{a}$ are not modified.) The PAC rules is *applicable* to pre-snapshots Σ, Σ' if $\Sigma \models \pi$ and $(\Sigma, \Sigma') \models \alpha'$. In this case, the result of *applying* the rule is Σ'', which is constructed from Σ' by changing the value of status attribute a according to $\odot a$. In this case we say that polarized attribute $\cdot a$ is *triggered* by the application of rule $(\pi, \alpha, \odot a)$.

A PAC rule is *applicable* to the sequence $\Sigma = \Sigma_0, \Sigma^e = \Sigma_1, \Sigma_2, \ldots, \Sigma_i$, if it is applicable to Σ, Σ_i; in this case the application of the rule adds Σ_{i+1} to the sequence where Σ_{i+1} is the result of applying the rule to Σ, Σ_i. ▮

The intuition for using the primed version α' rather than α in Definition 4.5 is as follows. Suppose that a sequence $\Sigma = \Sigma_0, \Sigma^e = \Sigma_1, \Sigma_2, \ldots, \Sigma_i$ of pre-snapshots has been constructed and rule $(\pi, \alpha, \odot a)$ is under consideration, with $\Sigma_0 \models \pi$. The sentry underlying α may refer to data attributes that were updated to form Σ_1 (including possibly $\mathsf{LatestIncEventType}$), and/or to status attributes that have changed when constructing $\Sigma_2, \ldots, \Sigma_i$. As is typical in rules-based systems, the condition α should be tested against the pre-snapshot that incorporates the effects of the previous updates. This is achieved by testing $(\Sigma, \Sigma') \models \alpha'$ rather than $(\Sigma, \Sigma') \models \alpha$.

4.3. Intuitions underlying B-steps: Toggle Once and Inertial

This subsection describes two key intuitions underlying the notion of B-steps, and provides illustrations of non-intuitive behaviors that will be prevented by the GSM well-formedness condition.

Recall that B-steps are intended to correspond to the smallest unit of business-relevant change in the state of an artifact. One intuitive principle of B-steps is called *Toggle Once*. This states that in a B-step (Σ, e, Σ'), if Σ' is constructed from (Σ, e, t) through the incremental application of PAC rules, then each status value attribute can change at most once during that construction. Note that if the incremental computation of a B-step did not satisfy Toggle Once, then a given status attribute might change values inside the B-step, but those changes would not be visible from the starting and ending snapshots of the B-step.

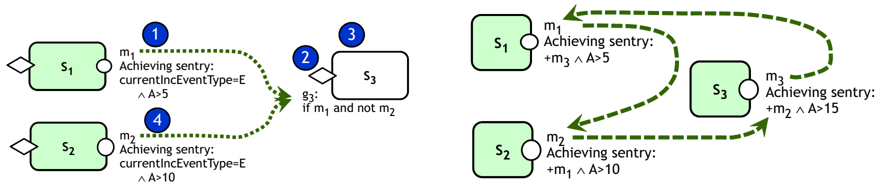
The Toggle Once principle is enforced by the use of prerequisites in the PAC rules (see Lemma 5.10). In the formalism, enforcing the Toggle Once principle has the advantage of preventing infinite cycles in the incremental computation of a B-step.

The second intuitive principle is called *inertial* (formalized in Definition 5.4 below). This states that if a status attribute changes during a B-step, then there should be a "justification" for that change that is visible by examining only the starting and ending snapshots of the B-step. Now, in the general case, the set of PAC rules of a GSM model Γ will involve a form of negation. As is well-known from logic programming and datalog [29], the presence of negation in rules can lead to outcomes that are not inertial. In the GSM operational semantics this will be avoided using an approach reminiscent of stratification as developed in those fields [4, 16]. In particular, the approach involves (i) requiring that a certain relation defined on the rules be acyclic, and then (ii) requiring that the order of rule firing comply with that relation. We note also that our framework satisfies a form of monotonicity because of the Toggle Once property. Specifically, in a B-step, once a status attribute changes, then it will not change again.

We now present an example of how unrestricted use of the PAC rules can lead to a non-inertial result.

Example 4.6. Consider Figure 7(a), and suppose that for some snapshot Σ we have that S_1 and S_2 are both open, that m_1 and m_2 are both false, that attribute $A = 20$, and that event e is to be processed. Suppose that the PAC rules are applied in the order suggested by the numbers in the figure, that is

1. milestone m_1 is achieved;
2. guard g_3 is triggered (since at this moment $m1$ is true and m_2 is false);
3. stage S_3 is opened;



(a) The need for ordering (b) The need for acyclicity

Figure 7: Illustration of potential non-intuitive behaviors (explained in Examples 4.6 and 4.7). Stages in green are open in the snapshot under consideration.

4. milestone m_2 is achieved;
5. (not numbered: stages S_1 and S_2 are closed)

Let Σ' be the result of these steps. Then Σ' is a snapshot. However, in Σ' we have that both m_1 and m_2 are true, or in other words, the condition of guard g_3 is not true in Σ'. Intuitively, there is not an apparent reason, looking only at Σ and Σ', as to why $S3$ has become open in Σ'. However, if using the ordering that will be imposed on the PAC rules, in this example the rules governing both m_1 and m_2 will have to be fired before any rules governing g_3 and S_3. ▮

A more subtle violation of inertial can arise if there are cycles, as illustrated next.

Example 4.7. Figure 7(b) shows a cycle of dependency between three milestones: achieving one can lead to achieving a second one, which can in turn lead to achieving the third one. Suppose now that in some GSM snapshot Σ each of S_1, S_2, and S_3 are open. One can imagine a B-step (Σ, e, Σ') where e has nothing to do with those stages, but where each of m_1, m_2, and m_3 are true in Σ' (and the corresponding stages are closed). In Σ' the change in value for each of m_1, m_2, and m_3 is “justified”, and so the “inertial” property is satisfied at a local level. However, intuitively it is unsatisfactory to permit this situation. The acyclicity conditon imposed on the PAC rules will prevent this kind of non-intuitive behavior. ▮

4.4. Polarized Dependency Graph and Well-formedness

The Polarized Dependency Graph (PDG) is intended to capture dependencies between the PAC rules in $\textsf{rules}_\Gamma$, and will be used to constrain the order of application of rules during the incremental construction of B-steps. This graph incorporates several aspects of the GSM semantics. The central intuition underlying the PDG is as follows: an edge from polarized attribute $\odot a$ to polarized attribute $\widehat{\odot}\widehat{a}$ is included in the PDG if consideration of PAC rules triggering $\widehat{\odot}b$ should be performed only after consideration of PAC rules triggering $\odot a$. In what follows, the definitions are presented, followed by some examples and intuitions.

Definition 4.8. The *polarized dependency graph* (*PDG*) of a GSM model Γ, denoted *PDG*(Γ), is defined as follows. For each status attribute a in Γ, we have two nodes $\langle +, a\rangle$ and $\langle -, a\rangle$. For each stage S and each of its guards φ, we have a node $\langle +, S.\varphi\rangle$.

When it is clear from the context, we abbreviate a node $\langle \odot, a\rangle$ by writing simply "$\odot a$". In the following description of the edges of the PDG, the antecedent α of a PAC rule is written as $\tau \wedge \gamma$, where τ is either empty, or has form $\mathsf{LatestIncEventYype} = E$ for some incoming event type E, or has the form $\odot a$ for some status attribute a and $\odot \in \{+, -\}$; and where γ involves no incoming even types or status events.

1. For each PAC-1 rule $\langle \neg active_S, \tau \wedge \gamma, +active_S\rangle$ in rules_Γ,
   + If $\widehat{\odot} b$ is a polarized status attribute occurring in τ, then include directed edge $(\widehat{\odot} b, +S.\varphi)$.
   + If a status attribute b occurs in γ, then include two directed edges $(+b, +S.\varphi)$ and $(-b, +S.\varphi)$.
2. For each guard φ of stage S:
   + add edge $(+S.\varphi, +active_S)$.
   + for each milestone m owned by S that does not occur in a top-level conjunct of form $\neg m$ in γ, add the edge $(+S.\varphi, -m)$. (These edges correspond to PAC-4.)
3. For each PAC rule $\langle \pi, \tau \wedge \gamma, \odot a\rangle$ from templates PAC-2 or PAC-3 in rules_Γ,
   + If $\widehat{\odot} b$ is a polarized status attribute occurring in τ, then include directed edge $(\widehat{\odot} b, \odot a)$.
   + If b is a status attribute occuring in γ, then add two edges $(+b, \odot a)$ and $(-b, \odot a)$.
4. For each PAC-5 rule $\langle \mathit{active}_S, +m, -\mathit{active}_S\rangle$ in rules_Γ, add edge $(+m, -\mathit{active}_S)$, and
5. For each PAC-6 rule $\langle \mathit{active}_S, -\mathit{active}_{S'}, -\mathit{active}_S\rangle$ in rules_Γ, add edge $(-\mathit{active}_{S'}, -\mathit{active}_S)$.

The *status-only polarized dependency graph*, denoted $\mathit{PDG}^s(\Gamma)$, is formed from $\mathit{PDG}(\Gamma)$ as follows. The nodes are $\{\odot a \mid a \text{ is a status attribute, and } \odot \in \{+, -\}\}$. Each edge of $\mathit{PDG}(\Gamma)$ that does not involve a guard is included. Also, for each pair of edges $(\odot a, +S.\varphi)$ and $(+S.\varphi, \widehat{\odot} b)$, add edge $(\odot a, \widehat{\odot} b)$. Finally, the transitive closure of $\mathit{PDG}^s(\Gamma)$ is denoted $\mathit{PDG}^{s*}(\Gamma)$. ▮

It is straightforward to verify that $\mathit{PDG}(\Gamma)$ is acyclic iff $\mathit{PDG}^s(\Gamma)$ is acyclic.

Definition 4.9. A GSM model Γ is *well-formed* if $\mathit{PDG}^s(\Gamma)$ is acyclic. ▮

Example 4.10. Figure 8 shows part of the PDG for the Engineering Requirements GSM model of Example 2.1, with a focus on the "Engineering Design" stage, its guards and its milestones. For each milestone (oval) and stage (rounded rectangle) there are two nodes, one corresponding to the positive polarity $(+)$ and the other to the negative polarity $(-)$. For each guard (diamond) there is only the positive polarity.

Consider first the node "+Design Suspended". Because of template PAC-5, a possible cause for "$-\mathit{active}_{\text{Engineering Design}}$" to become true (i.e., for "Engineering Design"

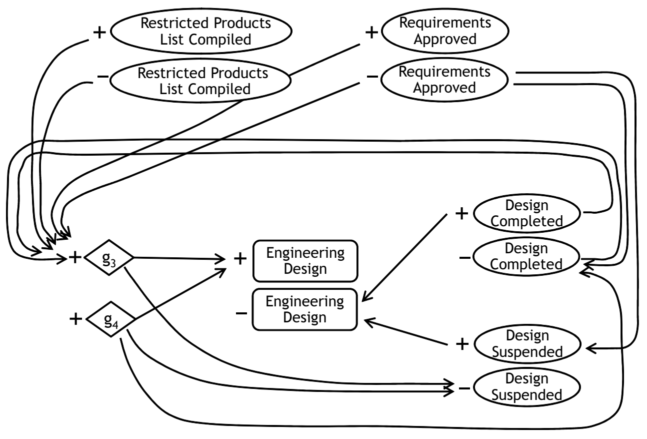


Figure 8: Part of PDG for GSM model of Engineering Requirements example. (Here rounded rectangles correspond to stages, ovals to milestones, and diamonds to guards. The '+' and '-' signs indicate the polarities associated with those nodes.)

to become inactive), is if "+ Design Suspended" becomes true. Intuitively, if computing a B-step through incremental application of the PAC rules, then rules concerning "$+$Design Suspended" should be evaluated before rules concerning "$-$Engineering Design". It is in this way that edges of the PDG indicate a dependency of the target of an edge on the source of that edge.

Consider now the guard node labeled "$+g_4$". Technically, this corresponds in Definition 4.8 to the node $\langle +, S.\varphi \rangle$, where S is the stage "Engineering Design" and φ is the sentry "LatestIncEventType = Request : ResumeEngineeringDesign." Because the sentry of g_4 involves only an incoming event, there are no edges in the PDG with target "$+g_4$".

According to Definition 4.8 an edge from $+g_4$ to $+active_{\text{Engineering Design}}$ is included, corresponding to the template PAC-1. Also, edges from $+g_4$ to "$-$Design Suspended" and "$-$Design Completed" are included, corresponding to the template PAC-4. Intuitively, in the general case, such edges are included because rules that can affect status attributes in a guard g of S should be considered before the PAC-1 based rule that uses g as antecedent can trigger $+active_S$, and also before the PAC-4 based rules that uses $+g$ as antecedent to invalidate milestones of S.

Consider now $+g_3$, which has associated sentry "ReqirementsApproved$\wedge$ Restrict$-$edProductsListCompiled $\wedge$ $\neg$DesignCompleted". Edges from both polarities of "Requirements Approved" to $+g_3$ are included because during incremental application of the PAC rules, the value of "Requirements Approved" should be stable before considering rules that have g_3 as antecedent. That is, rules that might trigger "$+$Requirements Approved" and rules that might trigger "$-$Requirements Approved" should both be considered before considering the value for $+g_3$. The analogous is true for "Restricted Products List Compiled" and "Design Completed".

Consider now out-going edges from $+g_3$. Similar to $+g_4$, edges are included from $+g_3$ to $+active_{\text{Engineering Design}}$ and "$-$Design Suspended". An edge from $+g_3$ to "$-$Design Completed" is not included. Intuitively, this relates to the formulation of

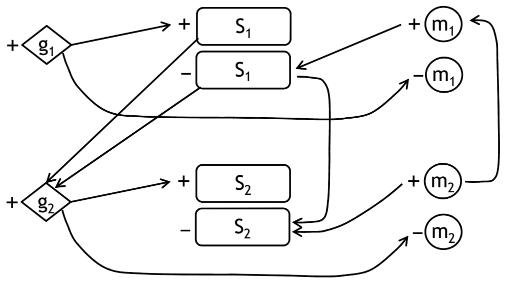


Figure 9: Illustration of an acyclic polarized dependency graph, where the corresponding non-polarized dependency graph is cyclic

template PAC-4 and the fact that $\neg$DesignCompleted is a top-level conjunct of g_3. Note that in an incremental application of PAC rules ordered according to the PDG, if g_3 were to fire, then necessarily "Design Completed" is false, and will not subsequently be modified. Thus, in connection with $\mathit{active}_{\text{Engineering Design}}$ and milestone "Design Completed," the invariant GSM-1 will be satisfied. ▮

The following example presents one motivation for using polarized status attributes in the PDG, rather than simply the status attributes.

Example 4.11. Figure 9 illustrates the PDG for a GSM model involving two stages S_i (with guard g_i and milestone m_i) for $i \in [1, 2]$, where S_2 is a child of S_1. The sole achieving sentry for m_1 is $+m_2$. (The sentries in the guards, and for achieving m_1, are not relevant to the example.) As shown in the figure, there are edges: $(+S_2.g_2, -m_2)$, $(+m_2, +m_1)$, $(+m_1, -S_1)$, $(-S_1, +S_2.g_2)$.

(The last edge is because active_{S_1} is a conjunct in the antecedent for the PAC-1 rule that corresponds to guard g_2.) If plain status attributes rather than polarized ones were used, this would yield a cycle. ▮

The PDG of a GSM model provides an ordering for application of the PAC rules that ensures an intuitively natural outcome for B-steps. The particular formulation of PDG (and of the PAC templates) described in the current paper was developed after working with many GSM models, some of which have been used in practice. If using the formulation of Definition 4.8, the PDGs of those GSM models are acyclic. The next two remarks consider alternative formulations for the PDG.

Remark 4.12. We develop here a formulation of PDG based on a simplified version of template PAC-4. In particular, suppose that PAC-4 is replaced by a template PAC-4', where PAC-4' applies to each stage S and milestone m where S owns m, and has prerequisite m, antecedent $+\mathit{active}_S$, and consequent $-m$. It would then be natural to modify Definition 4.8 as follows: for each stage S with guard φ and milestone m, remove the edge $(+S.\varphi, -m)$, and include an edge $(+S, -m)$. (That is, delete the second item in bullet (2.) of that definition, and use instead an edge from $+S$ to $-m$.)

This formulation of the PAC templates and the PDG is simpler than the ones of Definitions 4.4 and 4.8, respectively. However, the PDG created from Design-to-Order using this new formulation is not acyclic, because of the interplay of stage "Engineering Design" and milestone "Design Completed". ▮

The next formulation of PDG is more lenient than Definition 4.8, in the sense that it yields an acyclic graph for some GSM models that have cyclic PDGs

Remark 4.13. (***Event-relativized PDG.***) This formulation of PDG is based on the observation that B-steps are based on single incoming events. Given GSM model Γ and incoming event type E, we define the *event-relativized polarized dependency graph* (*ePDG*) of Γ for E, denoted *PDG*$_E(\Gamma)$, as follows. The set of nodes for *PDG*$_E(\Gamma)$ is the same as for *PDG*(Γ), except that an additional node $\langle +, E\rangle$ is included. The set of edges for *PDG*$_E(\Gamma)$ is constructed as follows:

1. Given a stage S with a guard φ that has E as triggering event, include edge $(+E, +S.\varphi)$.
2. For each milestone m with achieving sentry that has E as triggering event, include the edge $(+E, +m)$.
3. For each milestone m with invalidating sentry that has E as triggering event, include the edge $(+E, -m)$.
4. Recursively add edges that are (a) in *PDG*(Γ) and (b) have source reachable from edges already included into *PDG*$_E(\Gamma)$.

Intuitively, if a node $\odot a$ in *PDG*$_E(\Gamma)$ is reachable from $+E$, then it is possible that for some snapshot Σ of Γ and some event e of type E, the B-step resulting from incorporating e into Σ includes the status attribute change $\odot a$. (Technically, we are guaranteed only the converse of this property.)

Consider a GSM model with milestones m_1 and m_2, where m_1 has achieving sentry "LatestIncEventType = $E_1 \wedge m_2$" and m_2 has achieving sentry "LatestIncEventType = $E_2 \wedge m_1$." The PDG of Definition 4.8 for this model is cyclic, but the event-relativized PDGs for this model are acyclic. ▮

5. Equivalent Formulations of GSM Operational Semantics

This section presents the three formulations of the notion of B-step, and then proves their equivalence in the case of well-formed GSM models.

5.1. Incremental formulation

For this formulation, each PAC rule with consequent $\odot a$ is associated with the corresponding node of the graph *PDG*$^s(\Gamma)$ (e.g. $\langle \odot, a\rangle$ for a PAC rule with consequent $\odot a$). Since the PDG is acyclic, its topological sort provides a partial order $\prec$ on PAC rules. The main idea of the incremental semantics is that we apply the PAC rules in an order consistent with $\prec$.

In more detail, let Σ be a snapshot of a well-formed GSM model Γ and e an incoming event of type E that is applicable to Σ. Choose a topological sort $\odot_1 a_1, \odot_2 a_2, \ldots,$ $\odot_r a_r$ for *PDG*$^s(\Gamma)$. Construct the sequence

$$\Sigma = \Sigma_0, \Sigma^e = \Sigma_1, \Sigma_2, \ldots, \Sigma_n$$

inductively as follows. Suppose that Σ_i has been constructed from Σ_{i-1} by applying a rule with consequent $\odot_p a_p$, or that $i = 1$ in which case set $p = 0$. Consider the rules associated with $\odot_{p+1}a_{p+1}, \odot_{p+2}a_{p+2}, \ldots$ in sequence until some rule for some $\odot_q a_q$ is applicable to (Σ, Σ_i) (see Definition 4.5). Use that to form Σ_{i+1}. Continue until rules for all of the nodes in *PDG*$^s(\Gamma)$ have been considered.

Definition 5.1. Let Σ, Σ' be snapshots of a well-formed GSM model Γ and e an incoming event. Then (Σ, e, Σ') is an *incremental B-step* for Γ if Σ' is the result of applying PAC rules to Σ, Σ^e in the manner described above, for some topological sort of *PDG*$^s(\Gamma)$ ∎

Well-formedness guarantees the following important result.

Theorem 5.2. For a well-formed GSM model Γ, given a snapshot Σ and an applicable event e, there always exists a unique Σ' s.t. (Σ, e, Σ') is an incremental B-step. Further, Σ' satisfies GSM-1 and GSM-2.

The proof of this result and others are presented in Subsection 5.4 below.

5.2. Fixpoint formulation

The fixpoint formulation for the GSM semantics is analogous to the fixpoint characterization used in logic programming [29]. The formulation is based on two properties of triples (Σ, e, Σ'). In the following definitions, we assume that Σ, Σ' are snapshots of a well-formed GSM model Γ, and that e is an incoming event applicable to Σ. Intuitively, the first property states that Σ' must comply with all of the demands of the PAC rules.

Definition 5.3. Let Γ and (Σ, e, Σ') be as above. The triple (Σ, e, Σ') is *compliant* with respect to Γ if

- Σ' and Σ^e agree on all data attributes, and
- for each PAC rule $(\pi, \alpha, \odot a)$ in $\textsf{rules}_\Gamma$, if $\Sigma \models \pi$ and $(\Sigma, \Sigma') \models \alpha'$, then $\Sigma' \models \odot a$.

∎

Intuitively, the second property states that if a status attribute toggles between Σ and Σ', then that toggling must be "justified" by some PAC rule in connection with Σ and Σ'.

Definition 5.4. Let Γ and (Σ, e, Σ') be as above. The triple (Σ, e, Σ') is *inertial* with respect to Γ if for each status attribute a: if $\Sigma(a) \neq \Sigma'(a)$ then there is a PAC rule $(\pi, \alpha, \odot a)$ in $\textsf{rules}_\Gamma$ such that $\Sigma \models \pi$; $(\Sigma, \Sigma') \models \alpha'$; and $\Sigma' \models \odot a$. ∎

We pause now to consider template PAC-2, which has consequent of form $+m$ but prerequisite of form *active*$_S$ rather than $\neg m$.

Remark 5.5. Suppose that we attempt to create a template PAC-2' as an alternative to PAC-2, where PAC-2' has precedent $\neg m$ rather than *active*$_S$. Suppose that stage S is the owner of m. Because m may become true only if S is currently active, we need to include the condition $\textit{active}_S$ in the antecedent. So, PAC-2' has precedent $\neg m$, antecedent $\alpha = \textit{active}_S \wedge \varphi$, and consequent $+m$.

Suppose now that $\Sigma = \Sigma_0, \Sigma^e = \Sigma_1, \ldots, \Sigma_i$ is a partially constructed sequence of pre-snapshots according to the incremental formulation, except that PAC-2' is used in place of PAC-2. Suppose further that $\Sigma_0 \models \neg m$ and $(\Sigma_0, \Sigma_i) \models \alpha'$. This means that $\Sigma_i \models \textit{active}_S$, so the rule can be applied. Let us assume that the rule is applied, so that $\Sigma_{i+1} \models m$. Because of PAC-5, $\textit{active}_S$ will be negated in some subsequent micro-step in this B-step. Let Σ' be the result of the B-step. We then have $\Sigma' \not\models \textit{active}_S$. It follows that $(\Sigma, \Sigma') \not\models \alpha'$. This means that the PAC rule for $+m$ is not applicable to Σ, Σ'. But since $\Sigma \models \neg m$ and $\Sigma' \models m$, this means that (Σ, e, Σ') is not inertial (in the context of the modified PAC template). ∎

Returning to the main discussion, we now define the fixpoint formulation.

Definition 5.6. Let Γ and (Σ, e, Σ') be as above. Then the triple is a *fixpoint B-step* for Γ if

1. e is applicable to Σ,
2. for each data attribute a, $\Sigma'(a) = \Sigma^e(a)$
3. (Σ, e, Σ') is inertial and compliant with respect to Γ.

∎

5.3. *The Closed-Form Formulation*

The closed-form formulation of the GSM semantics is based on the observation that the properties of compliance and inertial can be captured in a first-order formula that refers to structures having the form (Σ, e, Σ'). The construction of the overall formula is reminiscent of constructions used for logic programming with negation, and in particular, when characterizing "negation as failure" [4, 16].

We sketch here how the formula is constructed. This formula will refer to attributes in Σ' as primed attributes. Also, for a PAC rule $(\pi, \alpha, \odot a)$ we write α in the form $\delta \wedge \gamma$ where δ is either empty or is a status change event. (If the rule is triggered by an incoming event, that is reflected in γ.) Finally, $\widehat{\delta}$ is the formula constructed from δ as follows: if δ has form $+a$ then set $\widehat{\delta}$ to be $\neg a \wedge a'$, and if δ has form $-a$ then set $\widehat{\delta}$ to be $a \wedge \neg a'$.

The main idea is to construct a logical formula from the set rules_Γ. Since many rules affect the same status attribute, for each attribute we first construct a formula that includes the effects of all such rules. We refer to all rules that have action $\odot a$ as $\textit{Cnsq}(\odot a)$.

For status attribute a, we define θ_{+a} to be

$$\begin{array}{l}((\neg a \;\wedge\; \bigvee_{(\pi,\delta\wedge\gamma,+a)\in \textit{Cnsq}(+a)}(\pi \;\wedge\; \widehat{\delta} \;\wedge\; \gamma')) \rightarrow a') \;\wedge \\ ((\neg a \;\wedge\; \bigwedge_{(\pi,\delta\wedge\gamma,+a)\in \textit{Cnsq}(+a)} \neg(\pi \;\wedge\; \widehat{\delta} \;\wedge \gamma')) \rightarrow \neg a')\end{array}$$

and define θ_{-a} to be

$$\begin{array}{l}((a \;\wedge\; \bigvee_{(\pi,\delta\wedge\gamma,-a)\in \textit{Cnsq}(-a)}(\pi \;\wedge\; \widehat{\delta} \;\wedge\; \gamma')) \rightarrow \neg a') \;\wedge \\ ((a \;\wedge\; \bigwedge_{(\pi,\delta\wedge\gamma,-a)\in \textit{Cnsq}(-a)} \neg(\pi \;\wedge\; \widehat{\delta} \wedge\; \gamma')) \rightarrow a')\end{array}$$

The intuition is that θ_{+a} considers the effect of all the rules that might result in a changed to $true$. Note how the first conjunct intuitively refers to the *compliance* requirement in the sense that it forces the attribute to transition to $true$ whenever any individual rule has prerequisite and antecedent both true. The second conjunct forces the *intertial* requirement by forcing the attribute to remain $false$ in case none of the rules can be "triggered". The same can be said for the θ_{-a}, which considers the rules that govern the transitions of the attribute to $false$.

For the whole system we then define:

$$\Theta_\Gamma = \Big(\bigwedge_{a \in \textit{Att}_{\textit{status}}} (\theta_{+a} \wedge \theta_{-a}) \Big) \ \wedge \ \Psi(e)$$

where $\Psi(e)$ is a formula (not defined here) that states that the data attributes of Σ' reflect the updates called for by e. (The condition Θ_Γ can also be extended to include the requirement that e is applicable to Σ.)

Definition 5.7. Let Γ and (Σ, e, Σ') be as above. Then the triple is a *closed-form B-step* for Γ if e is applicable to Σ and $(\Sigma, e, \Sigma') \models \Theta_\Gamma$. ∎

We can now state the main result.

Theorem 5.8. Suppose that Γ is a well-formed GSM schema, Σ, Σ' are snapshots of Γ, and e an incoming event applicable to Σ. Then the following are equivalent: (i) (Σ, e, Σ') is an incremental B-step, (ii) (Σ, e, Σ') is a fixpoint B-step, and (iii) (Σ, e, Σ') is a closed-form B-step.

Recall, as discussed in Introduction, that each of these three formulations has it own intrinsic value — the *incremental formulation* being useful primarily for direct GSM implementations (such as the one in the Barcelona GSM engine), the *fixpoint formulation* being of use in developing alternative (e.g., distributed) implementations and optimizations, and the the *closed-form formulation* being critical for verification purposes and for transferring existing verification results [7, 15, 12] to the GSM realm as we discuss in more detail in Subection 6.1.

The equivalence of the three specialized notions of B-step now allows us to formally define the unified notion of B-step.

Definition 5.9. Let Γ be a well-formed GSM model, Σ, Σ' be snapshots of Γ, and e an event applicable to Σ. Then (Σ, e, Σ') is a *B-step* for Γ if this triple is an incremental B-step for Γ (or equivalently, a fixpoint B-step for Γ or a closed-form B-step for Γ). ∎

5.4. Proofs

This subsection includes proofs for Theorems 5.2 and 5.8.

We begin with two lemmas. The first focuses on the relationship of the PAC rules and the Toggle Once principle.

Lemma 5.10. (**Toggle Once**) Let Γ be a well-formed GSM schema, Σ a snapshot of Γ, and e an incoming event applicable to Σ. Let $\Sigma = \Sigma_0, \Sigma^e = \Sigma_1, \Sigma_2, \ldots, \Sigma_n = \Sigma'$ be constructed according to the incremental semantics, using topological sort $\odot_1 a_1, \ldots,$

$\odot_r a_r$ of *PDG*$^s(\Gamma)$. Suppose that status attribute a changes value during the formation of Σ_i. Then a does not change value again in the sequence, i.e., for each $j \in [i..n]$, $\Sigma_j(a) = \Sigma_i(a)$.

Proof: Let S be a stage. Suppose that $\Sigma \models$ *active*$_S$. Then for each pre-snapshot $\widehat{\Sigma}$ we have that $(\Sigma, \widehat{\Sigma})$ violates the prerequisite of the PAC-5 and PAC-6 rules for S. Thus, if *active*$_S$ changes to false at some point in the sequence, then it cannot change back to true. A similar argument using PAC-1 can be used if $\Sigma \models \neg$*active*$_S$.

Suppose now $\Sigma \models m$. By GSM-1, $\Sigma \models \neg$*active*$_S$. Thus, for each pre-snapshot $\widehat{\Sigma}$, $(\Sigma, \widehat{\Sigma})$ violates the prerequisite of the PAC-2 rule for m. This implies that if m changes to false at some point in the sequence, then it cannot change back to true. Finally, suppose $\Sigma \models \neg m$. Thus, for each pre-snapshot $\widehat{\Sigma}$, $(\Sigma, \widehat{\Sigma})$ violates the prerequisites of rules PAC-3 and PAC-4. If m changes to true at some point in the sequence, it cannot change back to true. ∎

We now present a "stability" lemma and corollary for incremental constructions, that describe sufficient conditions for when a status attribute will remain fixed from an intermediate snapshot Σ_i until the end of the sequence.

Lemma 5.11. (**Stability**) Let Γ be a well-formed GSM schema, Σ a snapshot of Γ, and e an incoming event applicable to Σ. Let $\Sigma = \Sigma_0, \Sigma^e = \Sigma_1, \Sigma_2, \ldots, \Sigma_n = \Sigma'$ be constructed according to the incremental semantics, using topological sort $\odot_1 a_1, \ldots, \odot_r a_r$ of *PDG*$^s(\Gamma)$. If a rule with consequent $\odot a$ is applied to create Σ_i, and if $(\widehat{\odot} b, \odot a) \in$ *PDG*$^{s*}(\Gamma)$, then:

(a) if $(+b, \odot a)$ and $(-b, \odot a)$ are in *PDG*$^{s*}(\Gamma)$ then for each $j \in [i..n]$, $\Sigma_j(b) = \Sigma_{i-1}(b)$.

(b) if only one of $(+b, \odot a)$ or $(-b, \odot a)$ is in *PDG*$^{s*}(\Gamma)$, then for each $j \in [i..n]$, $(\Sigma, \Sigma_j) \models \odot b$ iff $(\Sigma, \Sigma_{i-1}) \models \odot b$.

Proof: Recall that a rule for $\odot a$ cannot be applied in the sequence until all rules are considered for polarized attributes $\odot' b$ where $(\odot' b, \odot a) \in$ *PDG*$^{s*}(\Gamma)$.

If both $(+b, \odot a)$ and $(-b, \odot a)$ are in *PDG*$^{s*}(\Gamma)$, then the rules for both $+b$ and $-b$ have been considered before Σ_i is constructed. If any such rules were applied, then the value of b has changed at or before Σ_{i-1}, and cannot change again because the Toggle Once Lemma 5.10.

Now consider the situation where only $(+b, \odot a) \in$ *PDG*$^{s*}(\Gamma)$. (The situation of $(-b, \odot a) \in$ *PDG*$^{s*}(\Gamma)$ is analogous.) There are two cases.
Case 1: $\Sigma \models b$. In this case there is no pre-snapshot $\widehat{\Sigma}$ such that $(\Sigma, \widehat{\Sigma}) \models +b$. Thus (Σ, Σ_j) does not model $+b$ for any $j \in [i-1..n]$.
Case 2: $\Sigma \models \neg b$. In this case, since $+b$ is considered before $\odot a$, it is considered before Σ_i is constructed. Thus, for each $j \in [i..n]$, $\Sigma_j(b) = \Sigma_{i-1}(b)$. This implies that for each $j \in [i..n]$, $(\Sigma, \Sigma_j) \models +b$ iff $(\Sigma, \Sigma_{i-1}) \models +b$. ∎

It is straightforward to verify the following.

Corollary 5.12. (**Stability**) Let Γ be a well-formed GSM schema, Σ a snapshot of Γ, and e an incoming event applicable to Σ. Let $\Sigma = \Sigma_0, \Sigma^e = \Sigma_1, \Sigma_2, \ldots, \Sigma_n = \Sigma'$ be constructed according to the incremental semantics, using topological sort $\odot_1 a_1, \ldots, \odot_r a_r$ of *PDG*$^s(\Gamma)$. Suppose that $\odot a$ is considered on Σ_i (whether or not it is applied to create Σ_{i+1}), and let $(\pi, \alpha, \odot a)$ be a rule for $\odot a$. Then for each $j \in [i..n]$, $(\Sigma, \Sigma_j) \models \alpha'$ iff $(\Sigma, \Sigma_i) \models \alpha'$.

The next two lemmas show that if Σ' is the result of an incremental construction from Σ and e, then Σ' is a fixpoint for Σ, e.

Lemma 5.13. (**Incremental is compliant**) Let Γ be a well-formed GSM model, Σ a snapshot for Γ, and e an applicable incoming event. If (Σ, e, Σ') is an incremental B-step then (Σ, e, Σ') is compliant with respect to Γ.

Proof: Assume that $\Sigma = \Sigma_0, \Sigma^e = \Sigma_1, \Sigma_2, \ldots, \Sigma_n = \Sigma'$ is constructed according to the incremental semantics, using topological sort $\odot_1 a_1, \ldots, \odot_r a_r$ of *PDG*$^s(\Gamma)$.

Consider a rule $(\pi, \alpha, \odot a)$, and suppose that $\Sigma \models \pi$ and $(\Sigma, \Sigma') \models \alpha'$. Suppose that in the incremental construction rules for $\odot a$ are considered on Σ_i (regardless of whether one of the rules is applied to create Σ_{i+1}). By the Stability Corollary 5.12, $(\Sigma, \Sigma_i) \models \alpha'$. Therefore if $\Sigma_i \not\models \odot a$, some rule with consequent $\odot a$ will be applied to create Σ_{i+1}. By the Toggle Once Lemma 5.10, $\Sigma_n \models \odot a$. ∎

Lemma 5.14. (**Incremental is inertial**) Let Γ be a well-formed GSM model, Σ a snapshot for Γ, and e an applicable incoming event. If (Σ, e, Σ') is an incremental B-step then (Σ, e, Σ') is inertial with respect to Γ.

Proof: $\Sigma = \Sigma_0, \Sigma^e = \Sigma_1, \Sigma_2, \ldots, \Sigma_n = \Sigma'$ and the topological sort be as in proof of Lemma 5.13. Suppose that $\Sigma \not\models \odot a$ and $\Sigma' \models \odot a$. We shall exhibit a rule $(\pi, \alpha, \odot a)$ such that $\Sigma \models \pi$ and $(\Sigma, \Sigma') \models \alpha'$. Since $\Sigma' \models \odot a$, the value of a was changed in some Σ_i. Let $(\pi, \alpha, \odot a)$ be the rule used. It follows that $\Sigma \models \pi$ and $(\Sigma, \Sigma_{i-1}) \models \alpha'$. By the Stability Corollary 5.12, $(\Sigma, \Sigma') \models \alpha'$. ∎

We next show that the result of an incremental construction satisfies the two invariants.

Lemma 5.15. Let Γ be a well-formed GSM model, Σ a snapshot for Γ, and e an applicable incoming event. If (Σ, e, Σ') is an incremental B-step then Σ' satisfies GSM-1 and GSM-2.

Proof: Assume that $\Sigma = \Sigma_0, \Sigma^e = \Sigma_1, \Sigma_2, \ldots, \Sigma_n = \Sigma'$ is constructed according to the incremental semantics. By Lemmas 5.13 and 5.14, (Σ, e, Σ') is both compliant and inertial.

Let stage S_1 have child S_2. In this case GSM-2 states that if S_1 is closed in Σ' then stage S_2 is closed in Σ'. Suppose that $\Sigma' \models \neg \textit{active}_{S_1}$. There are two cases to consider: $\Sigma \models \textit{active}_{S_1}$ or $\Sigma \models \neg \textit{active}_{S_1}$. In the first case, $(\Sigma, \Sigma') \models -\textit{active}_{S_1}$ and so by PAC-6 and compliance, $\Sigma' \models \neg \textit{active}_{S_2}$. In the second case, it follows from the Toggle Once Lemma 5.10 that for each $i \in [1..n]$, $\Sigma_i \models \neg \textit{active}_{S_1}$. By construction of PAC-1, no rule can be applied to change the value of $\textit{active}_{S_2}$ from false to true. Thus $\Sigma' \models \neg \textit{active}_{S_2}$ as desired.

Suppose now that stage S owns milestone m. In this case GSM-1 states that if S is open then m should be false. Suppose that $\Sigma' \models \textit{active}_S$. There are two cases to consider: $\Sigma \models \textit{active}_S$ or $\Sigma \models \neg \textit{active}_S$.

For the first case, suppose that $\Sigma' \models m$. Since Σ is a snapshot, Σ satisfies GSM-1 and in particular, $\Sigma \models \neg m$. Thus $(\Sigma, \Sigma') \models +m$. By compliance and PAC-5, $\Sigma' \models \neg \mathit{active}_S$, a contradiction.

Consider the second case. Then $(\Sigma, \Sigma') \models +\mathit{active}_S$. Suppose that $\Sigma \models -m$. The only rules with consequent $+m$ are from template PAC-2, and these have prerequisite active_S. Since this is false for Σ, no such rules will apply and so $\Sigma' \models -m$.

Suppose now that $\Sigma \models m$. Because (Σ, e, Σ') is inertial and $\Sigma' \models \mathit{active}_S$, there is some rule $(\neg\mathit{active}_S, \alpha, +\mathit{active}_S)$ that was applied to create the pre-snapshot Σ_i for some i during the construction of Σ. By the Stability Corollary 5.12, $(\Sigma, \Sigma') \models \alpha'$. There are two sub-cases: (a) $\neg m$ does not occur as a top-level conjunct in α or (b) it does. For (a), by PAC-4 there is a rule $(m, \alpha, -m)$. By compliance, $\Sigma' \models \neg m$ as desired. For (b), since the rule was applied on Σ_{i-1} to create Σ_i, $(\Sigma, \Sigma_{i-1}) \models \alpha'$. By the Stability Corollary, $(\Sigma, \Sigma') \models \alpha'$. Since $\neg m$ is a top-level conjunct in β, this implies that $\Sigma' \models \neg m$ as desired. ∎

We now demonstrate that if Σ' is obtained through an incremental construction from Σ and e, and if Σ'' is a fixpoint for Σ, e, then Σ' and Σ'' are identical. As a by-product we establish that the incremental construction is independent of the topological sort used, and that there is a unique fixpoint.

Lemma 5.16. Let Γ be a well-formed GSM model, Σ a snapshot for Γ, and e an applicable incoming event. If (Σ, e, Σ') is an incremental B-step and (Σ, e, Σ'') is a fixpoint B-step then $\Sigma' = \Sigma''$

Proof: Assume that $\Sigma = \Sigma_0, \Sigma^e = \Sigma_1, \Sigma_2, \ldots, \Sigma_n = \Sigma'$ is constructed according to the incremental semantics using topological sort $\odot_1 a_1, \ldots, \odot_r a_r$ of $\mathit{PDG}^s(\Gamma)$. Assume also that $\Sigma' \neq \Sigma''$. Of the status attributes a such that $\Sigma'(a) \neq \Sigma''(a)$, choose the one for which there is a polarity $\odot$ such that $\odot a$ is least in the topological sort among all $\widehat{\odot} b$ such that $\Sigma'(b) \neq \Sigma''(b)$.

Case 1: $\Sigma'(a) = \Sigma(a)$. We will show that Σ'' is not inertial. Suppose that in the construction of Σ', rules for $\odot a$ are considered at Σ_i. For each rule $(\pi, \alpha, \odot a)$, either $\Sigma \not\models \pi$ or $(\Sigma, \Sigma_i) \not\models \alpha'$. If $\Sigma \not\models \pi$ then none of the rules is satisfied by (Σ, Σ''). In this case, since Σ'' is inertial, $\Sigma''(a) = \Sigma(a) = \Sigma'(a)$, a contradiction.

Suppose now that $\Sigma \models \pi$. By the Stability Corollary 5.12, for each rule of form $(\pi, \alpha, \odot a)$ we have $(\Sigma, \Sigma') \not\models \alpha'$. Since $\odot a$ was chosen to be least among polarized attributes where Σ' and Σ'' differ, we also have that $(\Sigma, \Sigma'') \not\models \alpha'$. Again since Σ'' is inertial we reach a contradiction.

Case 2: $\Sigma'(a) \neq \Sigma(a)$. We will show that Σ'' is not compliant. There is some rule $(\pi, \alpha, \odot a)$ and some i such that this rule is applied to Σ_i to obtain Σ_{i+1}. Thus $\Sigma \models \pi$ and $(\Sigma, \Sigma_i) \models \alpha'$. By the Stability Corollary, $(\Sigma, \Sigma') \models \alpha'$. Since $\odot a$ was chosen to be least, Σ'' matches Σ' on each status attributes occurring in α. Thus $(\Sigma, \Sigma'') \models \alpha'$. Because (Σ, e, Σ'') is compliant, $(\Sigma, \Sigma'') \models \odot a$, and so $\Sigma''(a) = \Sigma'(a)$ afterall, a contradiction. ∎

Proof of Theorem 5.2. Let Σ be a snapshot of well-formed GSM model Γ and e an applicable event. First note that the incremental construction always succeeds, and by Lemmas 5.13 and 5.14, it creates a snapshot Σ' of Γ that is compliant and inertial. This implies that there is at least one fixpoint for Σ, e.

To see that there is only one incremental B-step for (Σ, e), suppose that there are two distinct snapshots Σ' and Σ'' such that (Σ, e, Σ') and (Σ, e, Σ'') are incremental B-steps. From Lemma 5.16, we see that these are both fixpoint B-steps as well. Since (Σ, e, Σ') is an incremental B-step and (Σ, e, Σ'') is a fixpoint B-step, Lemma 5.16 implies that $\Sigma' = \Sigma''$, a contradiction. ∎

Proof of Theorem 5.8. The proof of Theorem 5.2 above demonstrated that the notions of incremental B-step and fixpoint B-step are equivalent. Because of the construction of the closed form formulas, it is straightforward to show the equivalence of closed-form B-step with fixpoint B-step. ∎

6. Discussion

In this section we briefly consider four topics: (a) generalization of the verification techniques of [7, 15, 12] to apply to GSM; (b) variations of the GSM meta-model; (c) intuitions concerning sequences of B-steps; and (d) GSM implementation. Item (a) provides an important illustration of the value of the Equivalence Theorem 5.8, because it provides a bridge between the intuitive incremental formulation of the GSM operational semantics, and the succinct, logic-based closed-form formulation.

6.1. Generalization of the verification techniques to GSM

Here we briefly discuss how the existing verification results [7, 15, 12] can be applied to GSM. Notice that due to the equivalence of the three formulations of GSM operational semantics the generalized verification techniques can be directly applied to systems implementing the incremental as well as the fixpoint semantics.

References [7, 15, 12] develop verification results for a family of *sequential declarative artifact-based meta-models*. Similar to GSM, those models assume that artifacts have information models and lifecycles. These meta-models support *services*, that are essentially identical to the GSM tasks described above. In contrast with GSM where stages (and thus tasks) may run in parallel, in the sequential declarative meta-models the services operate in sequence. References [15, 12] also support the presence of a fixed external database that may be referred to in the pre- and post-conditions. The research reported there demonstrates decidability of questions of the form: do all runs of a sequential declarative artifact-based model satisfy a given LTL-FO formula.

The Closed Form formulation of the GSM semantics enables application of the proof techniques developed in [15, 12]. To see how, note that a run of a GSM model involves execution of B-steps in response to a series of incoming events. Each such B-step is characterized by the formula Θ_Γ (extended to include testing for applicability of an incoming event), and can be modeled as a service in the sense of [15, 12]. Also, the incoming events themselves can be modeled as the responses to other services calls. As a result, the closed-form formulation provides a direct way to translate GSM models into ‘sequential’ declarative artifact models. It follows that verification techniques and results in the spirit of [15, 12] can be extended to GSM.

6.2. Variations of the GSM meta-model

The larger GSM team has worked on a number of variations of the basic GSM meta-model studied in the current paper. As noted earlier, [22, 23] permit multiple artifact types and multiple artifact instances that can interact through the use of sentries.

A minor but convenient generalization of the GSM approach is to permit more than one event expression to be present in sentries. In this case, when putting a sentry into the form $\tau \wedge \gamma$ as in Definition 3.9, τ might involve a boolean combination of multiple incoming and internal event expressions. The results of the current paper can be generalized to this case.

Citation [40] studies a simplified GSM meta-model, in which milestones and stages are not tightly linked as in the current paper. In particular, milestones are not required to be linked to a stage, but rather can stand freely, either at the same level as the top-level stages, or nested within stages. Guards are used to govern when stages open, and "terminating sentries" govern when stages close. The PAC rules and notion of PDG can be simplified. Importantly, the simplified GSM meta-model can nevertheless simulate the tight linkage of milestones to stages as found in the current paper. The separation of milestones from stages was inspired in part by the initial approach to milestones taken in the IBM Case Manager product [47], and is also found in the emerging OMG standard for case management [8] (see Section 7).

In GSM as studied here, a stage may not have two occurrences that are running simultaneously. However, in some cases it is natural to use the same stage to perform work in parallel on multiple elements of a collection. For example, in business-to-business commerce it may be natural to process each line item of a purchase order using different occurrences of the same stage. It appears that the GSM meta-model could be extended with a form of *indexed stage execution* while still satisfying some analog of the Equivalence Theorem.

Finally, we note that in the abstract meta-model of this paper, concurrency control mechanisms are not provided. In particular, two tasks that write to the same data attribute might be executing in parallel, and the value written by the first task to complete might be overwritten when the second task completes. It would be natural to develop an approach for *business-level concurrency control*. This might require, for example, that an atomic stage would not be eligible to open if appropriate read- and write-locks on data attributes were not available. Develoiping this extension may require a generalization of the notion of B-step to allow for non-determinism which may arise from contention between atomic stages that may potentially open in a single B-step.

6.3. Sequences of B-steps

We now turn to situations where it makes intuitive sense to consider a cluster of B-steps as a single unit. In the formalism presented above, if an atomic stage contains a computational task (e.g., assigning one data attribute to equal another one), then this stage is opened in one B-step b_1 and is closed in some subsequent B-step b_2. Because the assignment is purely computational, it may make sense to have b_2 happen "immediately" after b_1. In practice, we define a *macro-B-step* to be a family of B-steps that starts with incorporation of an incoming event from the environment, and includes any subsequent B-steps stemming from computational actions. Macro-B-steps are not guaranteed to terminate, nor to be unique. We also note that in some corner cases, a change made by one B-step may be "undone" by another B-step in the same macro-B-step.

We also note that in some corner cases, a B-step may complete, and some PAC rule may be applicable to the result. As a simple example, suppose that S_1 and S_2 are two stages that are "siblings" in the hierarchy, with milestones m_1 and m_2, respectively. Let m_1 be triggered by incoming events of type E, let $+m_1$ be an achieving sentry of m_2, and let m_2 be a guard for S_1. Suppose that both stages are open and an event of type E arrives. The resulting B-step will include m_1 is achieved, S_1 is closed, and m_2 is achieved. The guard "m_2" of S_1 will not be fired, because the associated PAC rule has precedent $\neg \textit{active}_{S_1}$. Snapshots that are created by a B-step and for which a rule is immediately applicable are termed *orphans*. If an orphan does arise, then it seems natural to permit applicable rules to fire in the same macro-step as the initial B-step. It appears that GSM models that permit orphans do not commonly arise in practice. The problem of finding a useful syntactic characterization that guarantees that a model is orphan-free remains open at the time of writing.

6.4. GSM implementation

As mentioned in Introduction, a prototype engine, called Barcelona, is being developed to support experiments and implementations using GSM. Barcelona software supports execution as well as design of GSM models. Notice, that while in this paper (for ease of presentation) we have assumed a restriction that puts the focus on a single artifact instance of a single artifact type, Barcelona actually supports interactions of multiple artifact types and instances. Barcelona supports a simple graphical design editor, and captures the GSM models directly into an XML format. It is an outgrowth of the Siena system [10], that supports an artifact meta-model with state-machine based lifecycles. The expression language [28] used in Barcelona is an extension of the Object Constraint Language (OCL) [19]; it supports first-order logic constructs in a context of nested collection-based structures. The Barcelona engine has already been used for two pilots, including one [42] that supports a style of "knowledge- and data- intensive processes". The Barcelona implementation of GSM incorporates a number of practical capabilities, such as bindings between the variables of incoming messages and how their values are incorporated into appropriate artifact instances, handling of time outs and other failures, etc., that are not addressed in the current paper.

7. Related work

The GSM meta-model is a natural evolution from the earlier practical artifact meta-models [35, 10, 39], but using a declarative basis. It can be viewed as a natural evolution from the sequential declarative artifact-centric models of [7, 15, 12], extended to support modularity and parallelism within artifact instances. GSM draws on previous work on ECA systems [31], and develops a specialized variant useful for data-centric BPM.

There is a strong relationship between the artifact paradigm and Case Management [45, 14, 47]. In general, both approaches focus on conceptual entities that evolve over time, and support *ad hoc* styles of managing activities. In both approaches, there is a strong emphasis on conceptual entities that evolve as they move through a business. Both approaches make data a first-class citizen, and in particular call for maintaining an integrated view of all data that is business-relevant to a given entity (case) instance as it evolves. The artifact approach has been used in a variety of contexts for which case management is rarely if ever deployed, e.g., the use of multiple artifact types to support the management of financial "deals" [9], to manage "distributed enterprise services" [6], and to provide cross-silo visibility into the management of engineering changes in large-scale manufacturing. Also, while not being the focus of the current paper, the GSM meta-model provides richer declarative constructs for supporting the interaction of artifacts than typically found in the meta-models for Case Management [23]. Citation [42] provides various examples of such rich declarative interactions in GSM.

The work on Case Management that is arguably closest to the current paper is the Case Handling meta-model (called here "CH" for short) described in [45]. In both CH and GSM, models involve an information model and closely coupled process model. CH focuses on *activities* that are essentially the same as tasks of GSM. Unlike GSM, activities in CH have an explicit finite-state based lifecycle involving 6 states (*initial*, *ready*, *running*, *passed*, *skipped*, and *completed*). CH does not support grouping of activities into composite stages, nor does it support milestones as in GSM. In both CH and GSM, the process model is obtained by adorning units of work (activities in [45] and stages in GSM) with a form of pre- and/or post-conditions, and the operational semantics is based on ECA rules derived from them. In CH, the activities are arranged into a directed acyclic graph derived from the conditions, whereas in GSM there is an acyclicity requirement on the Polarized Dependency Graph (which has a node for both "positive" and "negative" versions of stages and milestones). A notion of roll-back is supported explicitly in CH, whereby if some activity A must be re-worked, then activities subsequent to A are also enabled for re-work. Although a detailed comparative analysis between CH and GSM has not been performed, it appears that with some minor extensions (including use of the event-relativized PDG mentioned in Remark 4.13), GSM can simulate essentially all of the characteristics of CH.

Recently, the perspective of Adaptive Case Management [41] has emerged; this permits more freedom than earlier case management approaches in how the processing of case instances is organized. Both GSM and Adaptive Case Management offer a spectrum of styles for managing the conceptual entities, from highly "prescriptive" to highly "descriptive". GSM includes a focus on the development of a precise mathematical definition for the operational semantics.

As noted in the Introduction, the leading industry consortium that is responding to the OMG call to standardize a Case Management Model Notation (CMMN) has adopted the core constructs of the GSM model, including guards, stages with hierarchy, milestones, and sentries [8]. There are some differences between GSM and the meta-model being developed there, including most notably (i) in [8] milestones and stages are more independent than in GSM (see Subsection 6.2); (ii) stages, tasks and milestones have finite-state machine based lifecycles (reminiscent of the Case Handling meta-model of [45]); and (iii) [8] includes rich mechanisms that enable case workers to modify the case model of case instances they are currently working with. An effort is underway to provide a variant of the GSM operational semantics for the meta-model of [8]. The consortium favors the GSM approach because it (i) can support the spectrum from prescriptive to descriptive in modeling behaviors, and (ii) it provides a formal foundation that is more declarative than classical, procedural approaches such as Petri nets.

Statecharts [20] have some features that resemble GSM, most notably the use of hierarchy for structuring clusters of work, and the support for parallelism. Fundamental differences between statecharts and GSM include: (a) GSM is fundamentally data centric whereas data is not central to the statechart meta-model (although it can be added); (b) statechart behavior is specified in a navigational, procedural manner, whereas GSM behavior is considerably more declarative, includes an explicit milestone construct, and permits far greater flexibility in representing the possible interleaving of parallel activities; and (c) although not explored in the current paper, artifact instance interaction is an important element of the GSM framework, and declarative mechanisms are provided to specify that interaction.

There is a relationship between the artifact approach and proclets [43]. Both approaches focus on factoring business operations into components, each focused on a natural portion of the overall operations, and where communication between components is supported in some fashion. Proclets use a Petri-net model to govern internal behavior and reaction to incoming events, and a message-based paradigm for interactions between the proclets. While proclets may maintain some data, the data cannot be shared except through the message-based interface. The artifact approach places more emphasis on data that is held and maintained by each component. GSM provides a declarative operational semantics, rather than one based on Petri nets. Although not studied in the current paper, in the case of multiple artifact types and instances, GSM permits declarative interactions between artifact instances. In particular, sentries of one artifact instance may incorporate tests against data, status attribute values, and/or status change events in other artifact instances [23].

Several other data-centric approaches to Business Process Management have been developed in recent years. Similar to the early artifact-centric models, Redding et. al. [36, 37] propose the *FlexConnect* meta-model where processes are organized as interacting business objects rather than as flows of activities. In *FlexConnect* lifecycles are defined as communicating finite state machines, with operational semantics defined by means of Coloured Petri nets. The states correspond to targeted activities to be performed during the overall lifecycle of an object. The objects can hold information, although it is not exposed to other objects or the external environment, as done in the business artifacts approach. Communication between objects can occur in sub-states of a state called gateways; these can be included at the start and end of any state. Flexibility is achieved by introducing patterns consisting of *coordination objects*, *job objects* and *referral objects*.

PHILharmonicFlows [26, 27] also defines a framework for object-aware process management which marries data with processes. The processes in PHILharmonicFlows are modeled on two levels: *micro processes* represent data and behavior of individual objects, while *macro processes* define interactions between the objects. The data model of objects is based on the relational data model, while the behavior is defined as a sophisticated combination of finite state machines and flows with well defined semantics and with transitions being dependent on data. The approach also defines access control and a mechanism for auto generated user forms.

Müller et. al. [33] propose the COREPRO framework for data-driven modeling of large process structures. The data structure types have associated with object lifecycles (defined as state transition systems), and relationships are used to characterize process structures. A domain specific data model consisting of object and relation types is defined at design time, and such a data model can then be instantiated to create specific data structures that serve as a definition of the run time process.

While the *FlexConnect*, PHILharmonicFlows and COREPRO approaches are all data-centric, there are significant differences compared to GSM. First, all these approaches are in essence based on variations of finite state machines or on a combination of finite state machines with flows. In this respect, these approaches are in essence more procedural with a navigational style semantics, compared to more declarative ECA style semantics of GSM. Next, GSM provides support for hierarchical organization of life-cycles by means of stages and sub-stages. PHILharmonicFlows introduces an explicit mechanism for separation of concerns between internal object behavior and object interactions by means of micro and macro processes, but the hierarchical organization of stages in GSM is different in the sense that it allows the hierarchical definition of behavior on the level of artifact instances. Finally, GSM enables rich parallelism in single instances,which is not supported by finite state machines.

Product-Based Workflow Design [38, 46] is a data-centric approach to workflow specification that focuses on definition of the information outcome (called Product Data Model, or PDM) of the workflow. In essence, PDM is a tree-like structure with nodes representing data elements and edges representing functional dependencies between the data elements. Actions are located on edges in-between data nodes, and generate new data values from the existing ones in a bottom-up manner. The PDM formalism represents how the final data outcome is calculated as well as the possible alternative ways of generating the outcome. Execution of the workflow is driven by the PDM structure and the possible choices between alternatives are derived from various action attributes, such as the price or the probability of failure. Compared to GSM, the Product-Based Workflow approach is specifically targeted at processes whose outcome can be represented in a tree like structure, which in turn can be used to drive the processes' execution. This approach imposes a rather specific operational semantics, which is in essence derived from the tree structure of PDM and from the availability of the data values. Artifact-centric processes and GSM, on the other hand, are focused on specification of generic processes where no such particular structure is assumed upfront.

Formal analysis of artifact-based business processes in various contexts has been reported in [17, 18, 7, 15, 12]. As noted in Subsection 6.1, unlike the GSM meta-model, all of these assume that the external tasks (services) are performed in a sequential fashion. Notably, [15] permits infinite data domains with order, and an underlying (static) database; and [12] extends to include arithmetic operations. Both works characterize bounds on expressive power that support decidability of LTL-FO properties.

The AXML Artifact model [2, 30] supports a declarative form of artifacts using Active XML [1] as a basis. Hierarchy based on the XML structure is used in AXML; this contrasts with the stage hierarchy in GSM. Automatic verification for AXML is studied in [3].

DecSerFlow [44] is a fully declarative business process language, in which the possible sequencings of activities are governed entirely by constraints expressed in a temporal logic. GSM does not attempt to support that level of declarativeness. In terms of essential characteristics, 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. The work in [21] falls in the same category as [44].

8. Conclusion

The Business Artifact paradigm provides a compelling data-centric approach for modeling and deploying business operations and processes, that is now incorporated into IBM products and service offerings. The Guard-Stage-Milestone (GSM) meta-model for artifacts represents a substantial extension of previous artifact meta-models, that supports a declarative style, parallelism within artifact instances, and modularity through hierarchical constructs. Based on the recent development of two substantial pilot projects using GSM, it appears that the core constructs of the GSM meta-model, and the essential aspects of the GSM operational semantics, are robust and will not undergo fundamental changes.

This paper provides the core mathematical foundations for GSM in an abstract setting. The paper introduces a well-formedness condition for GSM models that supports naturally arising patterns of dependencies and interrelationships between business operations that have been combined using declarative constructs, and helps to ensure key mathematical properties. The paper demonstrates the equivalence of three formulations for the GSM operational semantics, and describes how the result can be used to adapt verification results obtained for sequential artifact meta-models to the GSM context.

In this paper we have assumed a common restriction that puts the focus on a single artifact instance of a single artifact type and we did not consider interactions of multiple types and multiple instances. This restriction was motivated purely by making the presentation easier and, importantly, it does not fundamentally compromise the applicability of the results. Actually, citation [23] presents an extension of the meta-model described here that supports multiple artifact types, multiple artifact instances, and structured attribute values. (See also [24].) Also, the implemented execution engine Barcelona supports interactions of multiple artifact types and instances. The coordination between artifact instances is taking advantage of powerful declarative sentries as is illustrated for example in citation [42].

The development and results in this paper provide the foundation for a number of research and practical investigations into the use of declarative artifact-based frameworks. Importantly, an effort is currently underway to develop a variation of the GSM operational semantics for use with the emerging OMG standard for case management [8]. Some specific extensions of the GSM meta-model presented here include: (1) extension to multiple artifact types and instances (see [23, 24]); (2) simplification of the GSM meta-model by permitting a looser relationship between stages and milestones (see [40]); and (3) incorporation of "collection-indexed" stages, so that multiple occurrences of a stage can be executing simultaneously, where each occurrence corresponds to a different member of the indexing collection (e.g., performing an operation on each line-item in an purchase order). Some additional areas of investigation include: (4) incorporating business-level transactional guarantees; (5) incorporating roles, performers, teams, accountability, and delegation into an hierarchically organized lifecycle; (6) development of practical verification systems for GSM; (7) development of optimized implementations for GSM, including highly distributed, massively scalable ones; (8) the study of variations and customization of business processes; and (9) the development of a declarative approach for specifying transformations and integrations of legacy business processes into "views" of them.

Acknowledgements. The authors thank the many people in the extended Project ArtiFact team at IBM Research, the EU ACSI team [5], and others. Most notable among these are: David Cohn, Giuseppe De Giacomo, Manmohan Gupta, Riccardo De Masellis, Nirmit Desai, Alin Deutsch, Marlon Dumas, Fabiana Fournier, Hojjat Ghaderi, Fenno (Terry) Heath III, Stacy Hobson, Mark Linehan, Sridhar Maradugu, Renee Miller, Nanjangud C. Narendra, Anil Nigam, Mohammad Sadoghi, Biman Saha, Jianwen Su, Piwadee (Noi) Sukaviriya, Yutian (James) Sun, John Vergo, and Victor Vianu.

References

[1] S. Abiteboul, O. Benjelloun, and T. Milo. The Active XML project: An overview. *Very Large Databases Journal*, 17(5):1019–1040, 2008.

[2] S. Abiteboul, P. Bourhis, A. Galland, and B. Marinoiu. The AXML Artifact Model. In *Proc. 16th Intl. Symp. on Temporal Representation and Reasoning (TIME)*, 2009.

[3] S. Abiteboul, L. Segoufin, and V. Vianu. Static analysis of active XML systems. In *Proc. Intl. Symp. on Principles of Database Systems (PODS)*, 2008.

[4] K.R. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, Los Altos, CA, 1988.

[5] Artifact-centric service interoperation (ACSI) web site, 2011. `http:/acsi-project.eu/`.

[6] K. Bhattacharya, N. S. Caswell, S. Kumaran, A. Nigam, and F. Y. Wu. Artifact-centered operational modeling: Lessons from customer engagements. *IBM Sys. J.*, 46(4):703–721, 2007.

[7] K. Bhattacharya, C. E. Gerede, R. Hull, R. Liu, and J. Su. Towards formal analysis of artifact-centric business process models. In *Proc. Int. Conf. on Business Process Management (BPM)*, pages 288–304, 2007.

[8] BizAgi and Cordys and IBM and Oracle and SAP AG and Singularity (OMG Submitters) and Agile Enterprise Design and Stiftelsen SINTEF and TIBCO and Trisotech (Co-Authors). Proposal for: Case Management Modeling and Notation (CMMN) Specification 1.0, Feb. 2012. Document bmi/12-02-09, Object Management Group.

[9] T. Chao et al. Artifact-based transformation of IBM Global Financing: A case study. In *Intl. Conf. on Business Process Management (BPM)*, 2009.

[10] D. Cohn, P. Dhoolia, F.F. (Terry) Heath III, F. Pinel, and J. Vergo. Siena: From powerpoint to web app in 5 minutes. In *Intl. Conf. on Services Oriented Computing (ICSOC)*, 2008.

[11] D. Cohn and R. Hull. Business artifacts: A data-centric approach to modeling business operations and processes. *IEEE Data Eng. Bull.*, 32:3–9, 2009.

[12] E. Damaggio, A. Deutsch, and V. Vianu. Artifact systems with data dependencies and arithmetic constraints. In *Proc. Intl. Conf. on Database Theory (ICDT)*, 2011.

[13] E. Damaggio, R. Hull, and R. Vaculín. On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles. In *Intl. Conf. Business Process Mgmt. (BPM)*, 2011.

[14] H. de Man. Case management: Cordys approach, February 2009. `http://www.bptrends.com/deliver_file.cfm?fileType=publication&fileName=02-09-ART-BPTrends%20-%20Case%20Management-DeMan%20-final.doc.pdf`.

[15] A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *Proc. Intl. Conf. on Database Theory (ICDT)*, 2009.

[16] A. Van Gelder. Negation as failure using tight derivations for general logic programs. In *IEEE Symp. on Logic Programming*, pages 127–139, 1986.

[17] C. E. Gerede, K. Bhattacharya, and J. Su. Static analysis of business artifact-centric operational models. In *IEEE International Conference on Service-Oriented Computing and Applications*, 2007.

[18] C. E. Gerede and J. Su. Specification and verification of artifact behaviors in business process models. In *Proceedings of 5th International Conference on Service-Oriented Computing (ICSOC)*, Vienna, Austria, September 2007.

[19] Object Management Group. Object Constraint Language: OMG Available Specification, Version 2.0. `http://www.omg.org/technology/documents/formal/ocl.htm`, May 2006.

[20] D. Harel and A. Naamad. The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4), October 1996.

[21] T. Hildebrandt and R. R. Mukkamala. Distributed dynamic condition response structures. In *Pre-proceedings of Intl. Workshop on Programming Language Approaches to Concurrency and Communication Centric Software (PLACES 10)*, 2010.

[22] R. Hull, E. Damaggio, F. Fournier, M. Gupta, F. Heath III, S. Hobson, M. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, and R. Vaculín. Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *Proc. of 7th Intl. Workshop on Web Services and Formal Methods (WS-FM 2010), Revised Selected Papers*, Lecture Notes in Computer Science 6551. Springer-Verlag, 2010.

[23] R. Hull, E. Damaggio, R. De Masellis, F. Fournier, M. Gupta, F. Heath III, S. Hobson, M. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, and R. Vaculín. Business artifacts with guard-stage-milestone lifecycles: Managing artifact interactions with conditions and events. In *ACM Intl. Conf. on Distributed Event-based Systems (DEBS)*, 2011.

[24] R. Hull, E. Damaggio, R. De Masellis, F. Fournier, M. Gupta, F. Heath III, S. Hobson, M. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, and R. Vaculín. A formal introduction to business artifacts with guard-stage-milestone lifecycles, Version 0.8, May, 2011. Draft IBM Research internal report, available at `http://researcher.watson.ibm.com/researcher/view_page.php?id=1710`.

[25] S. Kumaran, P. Nandi, F.F. (Terry) Heath III, K. Bhaskaran, and R. Das. ADoc-oriented programming. In *Symp. on Applications and the Internet (SAINT)*, pages 334–343, 2003.

[26] Vera Künzle and Manfred Reichert. A modeling paradigm for integrating processes and data at the micro level. In Terry A. Halpin, Selmin Nurcan, John Krogstie, Pnina Soffer, Erik Proper, Rainer Schmidt, and Ilia Bider, editors, *Enterprise, Business-Process and Information Systems Modeling - 12th International Conference, BPMDS 2011, and 16th International Conference, EMMSAD 2011, held at CAiSE 2011, London, UK, June 20-21, 2011. Proceedings*, volume 81 of *Lecture Notes in Business Information Processing*, pages 201–215. Springer, 2011.

[27] Vera Künzle and Manfred Reichert. PHILharmonicflows: towards a framework for object-aware process management. *Journal of Software Maintenance*, 23(4):205–244, 2011.

[28] M. Linehan et al. GSM expression language, Version 1.0, January 28, 2011. IBM Research internal report, available on request.

[29] John. W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987.

[30] B. Marinoiu, S. Abiteboul, P. Bourhis, and A. Galland. AXART – Enabling collaborative work with AXML artifacts. *Proc. VLDB Endowment*, 3(2):1553–1556, Sept. 2010.

[31] D. R. McCarthy and U. Dayal. The architecture of an active data base management system. In *Proc. ACM SIGMOD Intl. Conf. on Mgmnt of Data (SIGMOD)*, pages 215–224. ACM Press, 1989.

[32] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[33] Dominic Müller, Manfred Reichert, and Joachim Herbst. Data-driven modeling and coordination of large process structures. In *OTM Conferences (1)*, pages 131–149, 2007.

[34] P. Nandi et al. Data4BPM, Part 1: Introducing Business Entities and the Business Entity Definition Language (BEDL), April 2010. `http://www.ibm.com/ developerworks/websphere/library/ techarticles/1004_nandi/1004_nandi.html`.

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

[36] G. Redding, M. Dumas, A.H.M. ter Hofstede, and A. Iordachescu. Modelling flexible processes with business objects. In *Proc. 11th IEEE Intl. Conf. on Commerce and Enterprise Computing (CEC)*, 2009.

[37] Guy Redding, Marlon Dumas, Arthur H. M. ter Hofstede, and Adrian Iordachescu. Transforming object-oriented models to process-oriented models. In Arthur H. M. ter Hofstede, Boualem Benatallah, and Hye-Young Paik, editors, *Business Process Management Workshops*, volume 4928 of *Lecture Notes in Computer Science*, pages 132–143. Springer, 2007.

[38] H.A. Reijers, S. Limam, and W.M.P. van der Aalst. Product-based workflow design. *Journal of Management Information systems*, 20(1):229–262, 2003.

[39] J.K. Strosnider, P. Nandi, S. Kumarn, S. Ghosh, and A. Arsanjani. Model-driven synthesis of SOA solutions. *IBM Systems Journal*, 47(3):415–432, 2008.

[40] Y. Sun, R. Hull, and R. Vaculin. Parallel processing for business artifacts with declarative lifecycles, 2012. Submitted for publication.

[41] Kieth D. Swenson. *Mastering the Unpredictable: How Adaptive Case Management will Revolutionaize the Way that Knowledge Workers Get Things Done*. Meghan-Kiffer Press, Tampa, FL, 2010.

[42] R. Vaculín, R. Hull, T. Heath, C. Cochran, A. Nigam, and P. Sukaviriya. Declarative business artifact centric modeling of decision and knowledge intensive business processes. In *The Fifteenth IEEE International Enterprise Computing Conference (EDOC 2011)*, pages 151–160. IEEE Computer Society, 2011.

[43] W. M. P. van der Aalst, P. Barthelmess, C.A. Ellis, and J. Wainer. Proclets: A framework for lightweight interacting workflow processes. *Int. J. Coop. Inf. Syst.*, 10(4):443–481, 2001.

[44] W. M. P. van der Aalst and Maja Pesic. Decserflow: Towards a truly declarative service flow language. In *The Role of Business Processes in Service Oriented Architectures*, 2006.

[45] W.M.P. van der Aalst, M. Weske, and D. Grünbauer. Case handling: a new paradigm for business process support. *Data Knowl. Eng.*, 53(2):129–162, 2005.

[46] Irene T. P. Vanderfeesten, Hajo A. Reijers, and Wil M. P. van der Aalst. Product-based workflow support. *Inf. Syst*, 36(2):517–535, 2011.

[47] W.-D. Zhu et al. Advanced Case Management with IBM Case Manager. Published by IBM. Available at `http://www.redbooks.ibm.com/redpieces/abstracts/sg247929.html?Open`.