A Formal Introduction to Business Artifacts with Guard-Stage-Milestone Lifecycles

Richard Hull1, Elio Damaggio2, Riccardo De Maselli3, Fabiana Fournier4, Mannmohan Gupta5, Fenno (Terry) Heath III1, Stacy Hobson1, Mark Linehan1, Sridhar Maradugu6, Anil Nigam1, Piwadee (Noi) Sukaviriya1, Roman Vaculin1

1IBM T.J. Watson Research Center, USA
2University of California, San Diego (elio@cs.ucsd.edu)
3University of Rome, La Sapienza(demasellis@dis.uniroma1.it)
4IBM Haifa Research Lab, Israel(fabiana@il.ibm.com)
5IBM Global Business Services, India (manmohan.gupta@in.ibm.com)
6Finsoft Consultants, Inc., USA (sridhar1@us.ibm.com)

Getting very close to Draft Version 0.8
May 16, 2011

Abstract

A promising approach to managing business operations is based on business artifacts (a.k.a. business entities with lifecycles). These are key conceptual entities that are central to guiding the operations of a business, and whose content changes as they move through those operations. An artifact type includes both an information model that captures, in either materialized or virtual form, all of the business-relevant data about entities of that type, and a lifecycle model, that specifies the possible ways an entity of that type might progress through the business by responding to events and invoking services, including human activities. While most previous work on artifacts has focused on the use of lifecycle models based on variants of finite state machines, two recent papers have introduced and studied the Guard-Stage-Milestone (GSM) meta-model for artifact lifecycles. GSM lifecycles are substantially more declarative than the finite state machine variants, and support hierarchy and parallelism within a single artifact instance.

This report presents in detail the formal operational semantics of GSM. This report is intended to serve as the authoritative source on the semantics of GSM; implementation efforts such as the Barcelona prototype engine may use this document as a reference manual. The flow of activity, and interactions between artifact instances, are supported both through testing of conditions against the artifact instances, and through events coming from an external environment or resulting from changes in artifact instances. The core of the operational semantics is based on the use of rules that are inspired by the Event-Condition-Action (ECA) rules paradigm. A key result here shows the equivalence of three different formulations of the GSM operational semantics. One formulation is based on incremental application

---

*This author partially supported by NSF grant IIS-0812578.
†Research by this author performed while visiting IBM Research, supported by NSF grant IIS-0812578.
‡The research leading to these results has received funding from the European Community’s Seventh Framework Programme FP7/2007-2013 under grant agreement number 257593 (ACSI).
§Work by this author performed while an employee of IBM India.
of the ECA-like rules, one is based on a pair of mathematical properties, and one is based on the use of first-order logic formulas.

This report is in the final stages of preparation. The current draft is being released informally to gather comments and suggestions before the report is formally released.


## 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\(^1\) Business Artifacts (a.k.a. “Business Entities with Lifecycles”), and has been studied by a team at IBM Research for several years [26, 20, 10, 25]. 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 entities of that type, and a lifecycle model, that specifies the possible ways that an entity 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 IBM team has recently introduced a declarative approach to specifying the lifecycles of business entities, using the Business Entities with Guard-Stage-Milestone Lifecycles meta-model\(^2\) (abbreviated “BEL[GSM]” or simply “GSM”) [17]. This technical report is intended to describe in a succinct and precise manner the core operational semantics for the GSM meta-model. In particular, three formulations are presented for the operational semantics and are shown to be equivalent. The development includes an emphasis on (a) how GSM supports declarative specification of interaction between business entities, and (b) how ECA-like rules are used to provide one formulation of the GSM operational semantics. The GSM meta-model will continue to evolve; this document describes version 1.0 of the meta-model.

As discussed in [17], the foundation of the research leading to GSM has been to create a meta-model for specifying business operations and processes that:

(i) Will help business-level stakeholders gain insight and understanding into their business operations;

(ii) Is centered around intuitively natural constructs that correspond closely to how business-level stakeholders think about their business;

(iii) Can provide a high-level, abstract view of the operations, and gracefully incorporate enough detail to be executable;

(iv) 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

---

1\(^1\) In some of the research literature related to the business artifacts paradigm in connection with IBM products or services the term ‘Business Entity’ has been used in place of ‘Business Artifact’. These terms refer to the same concept. Within IBM, the team has been using the term ‘entity’, because the term ‘artifact’ has a different, well-established meaning in the community of IT practitioners in the BPM space.

2\(^2\) 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.
(v) Provides a natural, modular structuring for specifying the overall behavior and constraints of a model of business operations in terms of ECA-like rules

(vi) 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.

Briefly, there are four key elements in the GSM meta-model: (a) Information Model for business entities, 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 based on triggering events and/or conditions. As will be seen, these provide a largely declarative framework for specifying a rich family of business-operations-motivated behaviors and interactions between entity instances.

This document is focused primarily on the technical building blocks used in the GSM meta-model (e.g., domain types, artifact types, artifact models, lifecycle models, etc.) and on of the operational semantics used by this meta-model. This semantics is based on a variation of the Event-Condition-Action (ECA) rules paradigm, and is centered around GSM Business steps (or B-steps), which focus on what happens to a snapshot (i.e., description of all relevant aspects of a GSM system at a given moment of time) when a single incoming event is incorporated into it. In particular, the focus is on what stages are opened and closed, and what milestones are achieved (or invalidated) as a result of this incoming event. 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 implementing GSM.

**Fixpoint:** This provides a concise “top-down” description of the effect of a single incoming event on an artifact snapshot. This 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 provides a characterization of snapshots and the effects of incoming events using what is essentially a first-order logic formula (extended here to work with nested sets). This permits the application of previously developed verification techniques to the GSM context. (The previous work [7, 14, 11] assumed that services were performed in sequence, whereas in GSM services and other aspects may be running in parallel.)

This document describes and motivates these three formulations of the semantics, and presents the proof of their equivalence. It also describes key properties of the incremental formulation that may be useful when developing optimized and/or distributed implementations of GSM.

Preliminary versions of the results presented in this report were published as [12, 18].

Although not a focus of the current document, we note that in the GSM framework, it is generally assumed that ad hoc queries can be made against the family of currently active artifact instances. These queries might be made by actors outside of a GSM system, and by artifact instances within the GSM system. In some cases, the queries might be subject to access controls. The ability to directly query the data held by artifact instances reflects the philosophical position that artifact instances correspond to business-relevant
entities that are moving through the business operations, that the data they hold is business-relevant, and as a result, it should be made available to appropriate actors and within the GSM system.

The GSM meta-model is being developed and studied in several ways. A concise, text-based programming language, called GSM-L, is being designed. (The core of this language, which is focused on the expression language used for the events and conditions used in guards and milestones, is available as [21].) A prototype engine, called Barcelona, is being developed to support experiments and implementations using GSM. (This supports a simple graphical design editor, and captures the GSM BOMs directly into an XML format.) Barcelona is an outgrowth of the Siena system [9], that supports an artifact meta-model with state-machine based lifecycles. The implemented meta-model must incorporate 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. The formal meta-model presented in this document makes a number of simplifying assumptions for clarity of exposition.

The IBM Research team has been working with the GSM meta-model for over two years. In addition to implementing the Barcelona prototype, the team has created several GSM models, including one that is being used for an IBM internal pilot application [29]. This gives strong reason to believe that the core constructs of the GSM meta-model, and the essential aspects of the GSM operational semantics, are robust and will not undergo significant changes. As discussed briefly in Section 5.5 there are some specific features that will be incorporated into meta-model in the coming months. Also, we expect some relatively superficial modifications and extensions to the GSM meta-model as the IBM team and larger community begin to work with applications modeled in GSM.

As briefly discussed in Section 6, there is a strong connection between GSM and the area of Adaptive Case Management [28], and also a connection between artifacts in general and the Petri-net based notion of proclets [30].

Organizationally, Section 2 gives an overview of the GSM meta-model through an example, including a discussion of how GSM supports declarative specification of interactions between business entities. Sections 3 and 4 introduce some of the formalism used to specify GSM models. Section 5 introduces and motivates the three formulations of the GSM operational semantics. Section 5 presents the definition of the well-formedness condition for GSM models, and presents the main results of the paper. Section 6 describes related work, and Section 7 offers brief conclusions.

2 Motivating Example: Requisition and Procurement Orders

This section describes the Requisition and Procurement Orders (RPO) scenario, and uses it to illustrate key features of the GSM meta-model. (The description of the GSM meta-model given here is brief; the reader may refer to [17], which illustrates the GSM meta-model using a different scenario, and provides more deeply the intuitive motivations for the GSM constructs.)

2.1 The Scenario

Briefly, in RPO a Requisition Order (or “Customer Order”) is sent by a Customer to a Manufacturer. The Requisition Order has one or more Line Items, which are individually researched by the Manufacturer to determine which Supplier to buy it from. The Line Items are bundled into Procurement Orders which are sent to different Suppliers.

---

3The authors thank Joachim (Jim) Frank of IBM for first introducing them to this problem scenario. We are using a simplified version of the scenario here.
A Supplier can reject a Procurement Order at any time before completion and shipment to the Manufacturer. In this case, the Line Items of that order must be researched again, and bundled into new Procurement Orders.

We focus here on the management of the orders, from Customer to Manufacturer and from Manufacturer to Suppliers. (We do not consider assembly of the part received from the Suppliers.) It is natural to model this scope of the Manufacturer’s operations using three artifact types, as follows.

**Requisition Order (RO):** Each RO instance will manage the overall operation of a single Requisition Order from initial receipt by the Manufacturer to delivery of the good(s) requested.

**Line Item (LI):** Each LI instance manages a single line item of a single requisition order. The main focus is to support the research for identifying which Supplier(s) to use, and to track the progress of the line item as it moves through research to being in a procurement order to arriving at the manufacturer.

**Procurement Order (PO):** Each PO instances manages a single procurement order, from when it is initially sent to a supplier to receipt of the goods or rejection by the supplier.

In the interest of simplicity, we do not consider error-handling in the scenario presented here. We note that typical error-handling will have business significance, and it can in general be modeled within the GSM framework.

### 2.2 Intuitive Framework for GSM Systems

In the general setting, a *Artifact Service Center (ASC)* is used to maintain a family of related artifact types and their associated instances. The ASC acts as a container and supports conventional SOA interfaces (using both WSDL and REST) to interact with an *(external) environment*. The most significant part of the environment for the discussion here is its ability to support 2-way service calls, which may be short-lived (as with most automated activities) or long-lived (as with most human-performed activities). The environment can also send 1-way messages into the ASC, and can request that the ASC create new artifact instances.

GSM, as with most BPM, case management, and workflow systems, is intended to support the *management* of business-related activities, but not support the details of executing those activities. Thus, most of the “actual work” in connection with a GSM model is typically performed by actors in the environment.

There are four primary components in the GSM meta-model, summarized here (further details given below).

**Information model:** Integrated view of all business-relevant information about an artifact instance, i.e., key conceptual entity, as it moves through the business operations.

**Milestone:** Business-relevant operational objective (at different levels of granularity) that can be achieved by an artifact instance. A milestone may be “achieved” (and become true when considered as a Boolean attribute) and may be “invalidated” (and become false when considered as a Boolean attribute).

**Stage:** Cluster of activity that might be performed for, with, and/or by an artifact instance, in order to achieve one of the milestones owned by that stage. A stage becomes “inactive” (or “closed”) when one of its milestones is achieved.

**Guard:** These are used to control whether a stage becomes “active” (or “open”).

---

5
Also very important in the GSM model is the following notion.

**Sentry:** This consists of a triggering event type and/or a condition. Sentries are used as guards, to control when stages open, and to control when milestones are achieved or invalidated. The triggering events may be incoming or internal, and both the internal events and the conditions may refer to the artifact instance under consideration, and to other artifact instances in the overall artifact system.

### 2.3 Drill-down into the RO artifact Type

Figure 1 illustrates the key components of the GSM meta-model through a sketch of the Requisition Order artifact type. This artifact type is firmly based on, and centered around, the information model, shown across the bottom. Here we see **Data Attributes**, which are intended to hold all business-relevant data about a given RO instance as it moves through the business. Speaking loosely, these attributes are generally filled up from left to right, although they may be overwritten. Also shown are two special data attributes, called *event occurrence bookkeeping* attributes, namely Most Recent Event Type and Most Recent Event Time, that are used to record information about the incoming event occurrence which is currently affecting or has most recently affected the artifact instance. These record, respectively, the type and the logical timestamp of this event. (As described below, the “logical” timestamp records the time when the event occurrence is incorporated into the artifact system.) Finally, the **Status Attributes** are illustrated there. These hold information about the current status of all milestones (true or false) and all stages (open or closed). These Status Attributes also hold, for each milestone and stage, the timestamp of the most recent change of status.
The upper portion of Figure 1 illustrates parts of the lifecycle model of the RO artifact type. **Milestones** are shown as small circles that are associated with stages. **Sentries**, i.e., specifications of a triggering event and/or a condition, are used to control whether a milestone is achieved (or invalidated; see below). A sentry is achieved and becomes true if an appropriate triggering event occurs and/or the condition becomes true. For example, one of the milestone achieving sentries of All Line Items ordered will become true if all Line Items associated with the RO instance are currently members of existing, non-rejected Procurement Orders. In this case the milestone is said to be achieved at that moment, and also the milestone, considered as a boolean attribute, is assigned the value true. Milestones may also be invalidated or “compromised”, and become false. Invalidations are also controlled by sentries. For example, All Line Items ordered is invalidated if a Procurement Order is rejected, in which case the Line Items in that Procurement Order will have to be researched again and one or more new Procurement Orders will have to be generated.

The rounded-corner rectangles correspond to **stages**, i.e., clusters of activity that may be performed in order to achieve one out of several milestones. (By construction, at most one milestone of a stage can be true at a time. Intuitively, each milestone of a stage corresponds to a distinct objective which might be achieved by the stage.) As illustrated by the stage Creating Procurement Orders, stages may be nested. Also illustrated are two **atomic** stages, namely Launching Line Items and Planning Proc. Orders. Both of these contain **tasks**, that involve activities that are modeled outside of the GSM model. As detailed below, there are two categories of task: to (a) invoke 2-way service call against the “environment” and (b) send 1-way message to the environment. (Sending a 1-way message to one or more other artifact instances is viewed as a special case of sending a 1-way message to the environment.) Figure 2 illustrates in more detail how artifact systems interact with the environment using tasks; see below.

If a stage is **active** or **open**, and if one of its milestones is achieved, then the stage becomes **inactive** or **closed**. Intuitively, this is because the overall motivation for executing a stage is to achieve one of its milestones.

**Guards** are used to control when stages become active. A guard is a sentry that is associated to a stage. If the sentry becomes true then the stage becomes active (or open).

Turning to sentries, two broad categories of events are considered. **Incoming events** correspond to events that can be generated by the environment. These include (a) the return calls from 2-way service calls that were invoked by some artifact instance; (b) 1-way messages from the environment, and (c) requests from the environment that a new artifact instance be created. (Messages from one artifact instance to another are also viewed as “incoming events”. Intuitively, this is because in the formal model, such messages are first sent to the ASC, and subsequently delivered to the receiving artifact instances.)

**Internal events** correspond to the changes in status of milestones (at the moment of being acheived or invalidated) and of stages (at the moment of being opened or closed; see below.) In the OCL-based expression language for GSM, the test for achieving [invalidating, respectively] a milestone \( m \) is written as \( m.onAchieved() \) \([m.onInvalidated()]\), and the test for activating [de-activiting] a stage \( S \) is written as \( S.onOpened() \) \([S.onClosed()]\).

### 2.4 Illustration of OCL-based language

We now illustrate how the OCL language, which is supported by IBM Research’s Barcelona prototype, is used to specify sentries. (See also [17, 21].) Consider first the sentry shown in Figure 1 associated with the milestone of Planning Proc. Order. Intuitively this will become true if the task inside that atomic stage terminates and sends a 2-way service call return back to the ASC, and the ASC maps this event to the calling artifact instance. The sentry associated to the milestone is specified as follows. (Assume that the service call return has type PlanOrdersReturn; also, the full payload of the return call is not detailed.)
For the next examples, the reader should refer to Figure 3 below, which shows some of the top-level stages of the lifecycle models for the three artifact types RO, LI, and PO. (The information models for the three artifact types are not shown.) We consider now the sentry shown in Figure 1 that can invalidate All Line Items ordered. The sentry will go true if any PO instances “owned” by the RO instance achieve the Rejected milestone.

\[ r.'Procurement Orders' \rightarrow \exists(p \mid p.'Rejected') \]

Here \( \rightarrow \) is a binary relation, involving the set of PO instances “owned” by the RO referred to by \( r \), and the condition that for at least one element \( p \) of that collection, the Rejected milestone is true.

As a final example, which shows the power of the sentries, we consider the guard Re-Order Line Items of Rejected Proc. Orders of Creating Procurement Orders. This will to true if (i) at least one LI instance was in a PO instance that was rejected and it has not since been re-ordered, and (ii) every LI instance with that property has re-achieved Potential Suppliers identified. To specify this guard we shall use two Boolean derived attributes in the information model of LI.

\[ \text{li.'ready to order'} \Rightarrow \]
\[ \text{li.'not ordered'} \land \text{li.'Potential Suppliers identified'} \]

\[ \text{li.'not ordered'} \Rightarrow \]
\[ \text{li.parentRO.'Purchase Orders' \rightarrow} \forall(p \mid \]
\[ p.'Line Items' \rightarrow \forall(l' \mid l'.ID < l.ID) \]
\[ \text{or} (p.'Line Items' \rightarrow \exists(l'' \mid l''.ID = l.ID \land p.'Rejected')) \]

Attribute not ordered will be true of an LI instance \( li \) if for each PO instance, either (a) the PO instance does not “own” \( li \) or (b) the PO instance does “own” \( li \) and that instance has milestone Rejected true. The guard can now be specified as

\[ p.'Line Items' \rightarrow \exists(l' \mid l'.ID < l.ID) \land p.'Line Items' \rightarrow \forall(l' \mid l'.ID = l.ID \land p.'Rejected')) \]

2.5 Interaction with the environment

Figure 2 illustrates at an intuitive level how 2-way service calls are invoked by artifact instances and run in the environment. The figure shows parts of one artifact instance snapshot that is part of the way through its execution. A stage is shown in green if it is currently open, and in pink if it has run at least once and is currently closed. Assume that this is the snapshot that arises after the Requisition Order has been in existence for some time, after the initial Procurement Orders have been launched, after at least one of those orders has been rejected, and just when Planning Proc. Orders opens. As illustrated in the figure, there are two 2-way service call occurrences that are already running in the environment. The act of opening the Planning Proc. Orders stage leads to the invocation of an occurrence of the 2-way service specified by the task within that stage. Note that this service may be long-running, e.g., if it is performed by a human. Eventually, it may be that the service occurrence will terminate, in which case a service call return message will be sent from the environment back to the ASC, which will in turn “route” the message to the instance that called the service occurrence. This will have the effect of closing the stage occurrence of Planning Proc. Orders. Alternatively, a different milestone might be achieved with the effect of closing the stage occurrence. This might arise, for example, if a manager determines that the overall activity should be stopped, e.g., because
the customer cancelled the order. In the formal model, the service occurrence might continue to run, but the service call return would be ignored by the artifact instance that called it. In the practical model, the ASC might send a message to the agent performing the service occurrence indicating that the service occurrence should be aborted. (Also, in a practical setting, it may be that the ASC and environment support a more rich style of interaction in connection with the service occurrence, so that a human performer can have an interactive engagement with selected attributes of the artifact information model; this is an area of active exploration.)

2.6 Declarative specification of artifact interactions

To conclude the discussion of the this example, we illustrate how the GSM constructs combine to permit the declarative specification of interactions between stages of a single artifact instance, and between related artifact instances. As noted above, Figure 3 shows some of the top-level stages of the lifecycle models for the three artifact types RO, LI, and PO.

We recall that the guards, milestone achieving sentries, and milestone invalidating sentries of one artifact type may refer to the information models of any artifact type. For example, the second guard of RO is based on the values of the milestone attribute Potential suppliers identified of LI instances that where participating in a PO instance that got rejected. This illustrates some of the richness of this declarative mechanism for specifying interactions between artifact instances of different types.

Recall that a GSM Business step (B-step) corresponds to the incorporation of a single incoming event into a GSM system, including all implied achieving/invalidating of milestones and openings/closings of stages. Using informal diagramatic conventions, the colored, dashed lines in Figure 3 illustrate three possible B-steps, which show how particular how an incoming event can trigger internal events and changes to an artifact instance.

For the first example, consider the two blue arrows (with short dashes) in the LI lifecycle. In this example, the milestone Initialized is triggered by an incoming event (e.g., that the automated process that checks certain validity conditions about the line item has completed). Also, suppose that the guard labeled Ready to Research has no condition, and has as event that the milestone Initialized has been achieved. In the
Creating Procurement Orders

Initiate Req. Order

Planning Proc. Orders

Launching Line Items

Launching & Sending Proc. Orders

All Line Items ordered

Rec. Order cancelled

Generating Report

Assembling

Line Item created

Initializing

Initialized

Ready for Research

Researching

Potential Suppliers Identified

 Req. Order created

Initializing

Sent to Manu.

Received by Manu.

Building

All Built

Rejected

Figure 3: Interactions between artifact types
notation used in the current paper, this is written as +\text{Initialized} (here \( l \) is the “context variable” for the type \( LI \), and is used to refer to “self” in this case). In the B-step where milestone \text{Initialized} is achieved, the guard \text{Ready to Research} will become true and the stage Researching will be opened.

For the second example, consider the five green arrows (with long dashes) eminating from the PO milestone \text{Rejected}. These correspond to four kinds of actions that might occur in a B-step if some PO instance \( p \) achieves the milestone \text{Rejected}. In particular, the milestone \text{All Line Items ordered} for the RO instance that “owns” \( p \), if it is currently true, will be invalidated. This corresponds to the intuition that there are now some LI instances that must be ordered from some other Supplier. Also, the milestone for successful completion of Planning Proc. Orders is invalidated whenever \text{All Line Items ordered} is invalidated. Turning to the impact on LI, for each LI instance \( l \) that is “owned” by \( p \), the guard owning Proc. Order rejected of the Researching stage is triggered, and this stage is re-opened. And finally, in order to maintain a GSM invariant (see Subsection 3.3), the milestone Potential Suppliers identified for \( l \), if true, is invalidated.

For the third example, consider the four purple arrows (with long-short dashes), starting with the one that connects the Potential Suppliers identified milestone of LI with the guard Re-Order Line Items of Rejected Proc. Orders of Creating Procurement Orders in RO. That guard has no explicit triggering event, and its condition states, basically, “for each PO instance \( p \) that achieved \text{Rejected}, each LI instance “owned” by \( p \) has achieved Potential Suppliers identified”. Speaking intuitively, if this condition becomes true, then each of the LI instances from a rejected PO has been researched, and so a new round of PO planning can be initiated. In particular, as suggested by the arrow from the guard to the interior of Creating Procurement Orders, if the guard becomes true then this stage will open. Furthermore, in this example the guard of substage Planning Proc. Orders will become true once its parent stage is open, and so the substage will also open.

In practice, if the RO has many associated PO’s, and PO’s are being rejected every few days, then this condition may not become true for a very long time. In this situation, a third guard might be added to Creating Procurement Orders, that enables a manager to explicitly request that the stage be re-opened in order to process the LI instances that have already been researched. (What if this stage is already open, along with substage Planning Proc. Orders, and then additional LI instances achieve Potential Suppliers identified? In practice, if the task supporting Planning Proc. Orders is performed by humans, then the performer determining the new PO’s could be asked to incorporate the new LI’s into this process. In particular, then, this would be done while Planning Proc. Orders was open. If the task supporting Planning Proc. Orders is automated, then it probably happens quickly. In this case, a new guard can be added to Planning Proc. Orders, so that a manager will have the ability to request a re-running of that substage.)

When using a guard with no explicit triggering condition, it is important to ensure that the guard does not become true inappropriately. As a simple example, suppose that \( g \) is the guard of Launching Line Items in Figure 1, and the \( m \) is the milestone. In principle, the stage should open when the parent stage Creating Procurement Orders opens, and so \( g \) could be simply \text{true}. However, as indicated in the figure, the intention is that the stage Launching Line Items should occur just once, and not repeatedly. This can be achieved by using not \( p.m \) as the guard \( g \). This device, of including into a guard as conjuncts the negations of milestones of a stage, is a useful pattern when using guards without triggering conditions.

### 2.7 Adding Flowchart Arrows to artifact Lifecycle Models

We briefly mention that conditional flowchart arrows can be added as a formal part of an artifact lifecycle model To illustrate by example, recall that the guard \text{Received by Manu} includes as triggering event +\text{p.’Sent to Manu’}. If the condition of this guard is \( \varphi(p) \), then the guard can be replaced (or visualized) as an arrow, labeled by \( \varphi(p) \), from the milestone Sent to Manu to the stage Building.
3 GSM Models

This section describes the information models for artifact types and GSM models. The specification of lifecycle models is given in the next section.

3.1 Domain types

In order to speak precisely about artifact information models and the data held by them, we quickly introduce the underlying family of data domains and types used.

We assume the existence of the following pairwise disjoint countably infinite sets: \( \text{TYPE}_p \) of primitive types, \( \text{TYPE}_{\text{ART}} \) of artifact type names, \( \text{ATT} \) of attributes, \( \text{ID}_R \) of (artifact) identifiers for each type \( R \in \text{TYPE}_{\text{ART}} \). If \( \rho \in \text{ID}_R \), then we say that \( \rho \) has type \( R \).

We assume that \( \text{TYPE}_p \) includes Boolean, and also two specific types: \( \text{IncEVENT} \) that ranges over incoming event types (these will be detailed further in Section 4); and \( \text{TIMESTAMP} \) that ranges over logical timestamps (a linearly ordered set that corresponds intuitively to the set of possible times at which an individual incoming event in brought into a ASC).

If \( n \geq 1 \), then \( \langle A_1:T_1, ..., A_n:T_n \rangle \) is an \( n \)-ary tuple type, if each \( A_i \) is a distinct attribute in \( \text{ATT} \) and each \( t_i \) is in \( \text{TYPE}_p \cup \text{TYPE}_{\text{ART}} \) (i.e., either a primitive type or an artifact type). For a tuple type \( T \), the relation type over \( T \) is denoted as \( \{T\} \). Let \( \text{TYPE}_{\text{Rel}} \) be the set of all relation types.

It is useful to model situations where an attribute value of an artifact instance, or of a tuple occurring in a relation, is undefined. To this end each domain of values is extended with the special symbol “⊥”. The domain of each type \( T \), denoted as \( \text{DOM}(T) \), is defined as follows: (1) if \( T \in \text{TYPE}_p \) is a primitive type, the domain \( \text{DOM}(T) \) is some known set of values (integers, strings, etc.) along with \( \bot \); (2) if \( T \in \text{TYPE}_{\text{ART}} \) is an artifact type, \( \text{DOM}(T) = \text{ID}_T \cup \{\bot\} \); (3) if \( T \) is tuple type \( \langle A_1:T_1, ..., A_n:T_n \rangle \) then \( \text{DOM}(T) \) is \( \{\bot\} \) union the set of tuples over \( A_1:T_1, ..., A_n:T_n \), that is, the set of total functions \( \mu \) with domain \( A_1, ..., A_n \) such that \( \mu(A_i) \in \text{DOM}(T_i) \) for each \( i \in [1..n] \) (where some of the \( A_j \)’s may map to \( \bot \)); and (4) \( \text{DOM}(\{T\}) \) for tuple type \( T \) is defined to contain \( \bot \) and all finite sets over \( \text{DOM}(T) \).

A type is permitted for an artifact attribute if it is an element in the union \( \text{TYPES}_{\text{permitted}} = \text{TYPE}_p \cup \text{TYPE}_{\text{ART}} \cup \text{TYPE}_{\text{Rel}} \).

In the formal GSM meta-model, we use an extension of First-Order Logic (FOL) that supports (i) multiple sorts; (ii) objects with structure record of scalars and collection of record of scalars; (iii) support for null values by viewing all data values as ordered pairs, with first coordinate a Boolean indicating whether the value is the \( \bot \) or not; (iv) the “dot” notation to form path expressions (both into record types and to follow links based on artifact IDs), and a binary predicate \( \in \) to test membership in a collection; and (v) quantification over both collection types in artifact instances and over the full domain of currently active instances of an artifact type. Given that the artifact pre-snapshots are uniquely identified by their IDs, it is well-known that expressions in this extended FOL can be transformed into equivalent expressions in classical FOL.

(In the practical GSM meta-model, arbitrarily deep nesting of the relation construct is permitted. Although not done here, such nesting can also be supported in the formal model. In this case, it is convenient to assume that at each level the relation has a set of scalar attributes as a key, as in the nested relational model with Projected Normal Form (PNF) or as in the Verso data model; see [3].)
3.2 Artifact types and GSM models

This subsection introduces the notions of artifact types, which provide the structure for instances of business entities, and GSM models, which provide the structure for a family of related business entity types and their instances.

Definition: A artifact type has the form \((R, x, Att, Typ, Stg, Mst, Lcyc)\) where the following hold.

- **R** is the name of the GSM type.
- **x** is a variable that ranges over the (IDs of) instances of **R**. This is called the context variable of **R** and is used in the logical formulas in \(\mathcal{L}\).
- **Att** is the set of attributes of this type. **Att** is partitioned into the set **Att\text{data}** of data attributes and **Att\text{status}** of status attributes (see below).
- **Typ** is the type function for the data attributes, i.e., \(Typ:Att \to \text{TYPES}_{\text{permitted}}\).
- **S** is the set of stage names, or simply, stages.
- **M** is the set of milestone names, or simply, milestones.
- **L** is the lifecycle model of this artifact type (defined below).

The following must also hold for \((R, x, Att, Typ, Stg, Mst, Lcyc)\).

1. The sets **Att**, **S**, **M** are pairwise disjoint.
2. The set **Att**, and in particular **Att\text{data}**, includes the following special attributes.
   - (a) **ID**, which holds the unique, immutable identifier of the artifact instance. (This attribute can never take the null value \(\bot\).)
   - (b) **mostRecEventType** of type **IncEVENT**. Intuitively, for a given instance \(\sigma\) of type **R**, this attribute will hold the type of the incoming event occurrence relevant to \(\sigma\) that is “currently” being processed, or that has most recently been processed.
   - (c) **mostRecEventTime** of type **TIMESTAMP**. Intuitively, for a given instance \(\sigma\) of type **R**, this attribute will hold the logical timestamp of the processing of the incoming event occurrence relevant to \(\sigma\) that is “currently” being processed, or that has most recently been processed.
3. For each milestone \(m \in M\), there are two attributes in **Att\text{status}**, namely:
   - A milestone status value attribute, denoted as \(m\), of type Boolean.
   - A milestone toggle time attribute, denoted as \(m_{\text{mostRecentUpdate}}\), of type **TIMESTAMP**.
Intuitively, attribute \(m\) indicates whether milestone \(m\) is currently true or false, and the attribute \(m_{\text{mostRecentUpdate}}\) indicates the most recent logical timestamp in which the value of \(m\) changed.
4. For each stage \(S \in S\), there are two attributes in **Att\text{status}**, namely:
   - A stage status value attribute, denoted as \(active_S\), of type Boolean.
   - A stage toggle time attribute, denoted as \(active_S^{\text{mostRecentUpdate}}\), of type **TIMESTAMP**.
Intuitively, attribute $active_S$ indicates whether stage $S$ is currently active or inactive, and the attribute $active_{mostRecentUpdate}^S$ indicates the most recent logical timestamp in which the value of $active_S$ changed.

An artifact type $(R, x, Att, Typ, Stg, Mst, Lcyc)$ is often referred to using simply its name $R$. We use $ID_B$ to denote the type of IDs of artifact instances of $R$.

The structure of artifact type lifecycle models is defined next. Several details about lifecycle models are deferred until Section 4 below.

**Definition:** Let $(R, x, Att, Typ, Stg, Mst, Lcyc)$ be an artifact type. The lifecycle model $L$ of $R$ has structure $(Substages, Task, Owns, Guards, Ach, Inv)$ and satisfies the following properties.

- **Substages** is a function from $S$ to finite subsets of $S$, such that the relation $\{ (S, S') \mid S' \in Substages(S) \}$ creates a forest. The roots of this forest are called top-level stages, and the leaves are called atomic stages. A non-leaf node is called a composite stage.

- **Task** is a function from the atomic stages in $S$ to the set of possible tasks over $R$ (defined in Section 4).

- **Owns** is a function from $S$ to finite, non-empty subsets of $M$, such that $Owns(S) \cap Owns(S') = \emptyset$ for $S \neq S'$. A stage $S$ owns a milestone $m$ if $m \in Owns(S)$.

- **Guards** is a function from $S$ to finite, non-empty sets of sentries (defined in Section 4).

- **Ach** is a function from $M$ to finite, non-empty sets of sentries. For milestone $m$, each element of $Ach(m)$ is called an achieving sentry of $m$.

- **Inv** is a function from $M$ to finite sets of sentries. For milestone $m$, each element of $Inv(m)$ is called an invalidating sentry of $m$.

If $S \in Substages(S')$, then $S$ is a child of $S'$ and $S'$ is the parent of $S$. The notions of descendant and ancestor are defined in the natural manner.

We now have:

**Definition:** A GSM model is a set $\Gamma$ of artifact types with form $(R_i, x_i, Att_i, Typ_i, Stg_i, Mst_i, Lcyc_i), i \in [1..n]$, that satisfies the following:

1. **Distinct type names:** The artifact type names $R_i$ are pairwise distinct.

2. **No dangling type references:** If an artifact type $ID_B$ is used in the artifact type $R_i$ for some $i \in [1..n]$, then $R = B_j$ for some (possibly distinct) $j \in [1..n]$.

(As a convenience, we also assume that all of the context variables are distinct.)

Let GSM model $\Gamma$ be as above. As will be seen, the sentries of guards and milestones in one GSM type $R_i$ of $\Gamma$ may refer to the values of attributes in $Att_j$ for type $R_j$ for any $j \in [1..n]$, not just for $j = i$. 

14
3.3 (Pre-)Snapshots and Instances

The notions of “snapshot” and “instance” for both artifact types and GSM models are now introduced. Structural aspects of these notions are captured using the auxiliary notion of “pre-snapshot”. While our focus is on snapshots and instances, we shall use pre-snapshots to describe the incremental construction of a new GSM snapshot resulting from the impact of an incoming event on an existing GSM snapshot.

Let $\Gamma$ be a GSM model, and let $(R, x, \text{Att}, \text{Typ}, \text{Stg}, \text{Mst}, \text{Lcyc})$ be an artifact type in $\Gamma$.

In this context, an artifact instance pre-snapshot of type $R$ is an assignment $\sigma$ from $\text{Att}$ to values, such that for each $A \in \text{Att}$, $\sigma(A)$ has type $\text{Typ}(A)$. (Note that $\sigma(A)$ may be $\bot$ except for when $A = \text{ID}$.)

Let $\sigma$ be an artifact instance pre-snapshot and let $\rho = \sigma(\text{ID})$. If understood from the context, we sometimes use $\rho$ to refer to the pre-snapshot $\sigma$. In this case, if $A$ is an attribute of $R$, then $\rho.A$ is used to refer to the value of $\sigma(A)$.

The relationship of stages and milestones is fundamental to the GSM meta-model. Core aspects of this relationship are captured in the following three GSM Invariants, which apply to artifact instance pre-snapshots. Let $\sigma$ be an instance pre-snapshot of artifact type $R$ with ID $\rho$. The GSM Invariants are specified as follows.

**GSM-1: Milestones false for active stage.** If stage $S$ owns milestone $m$, and if $\rho.\text{active}_S = \text{true}$, then $\rho.m = \text{false}$.

**GSM-2: No activity in closed stage.** If stage $S$ has substage $S'$, and $\rho.\text{active}_S = \text{false}$, then $\rho.\text{active}_{S'} = \text{false}$.

**GSM-3: Disjoint milestones.** If stage $S$ owns distinct milestones $m$ and $m'$, and $\rho.m = \text{true}$, then $\rho.m' = \text{false}$.

(The third invariant is typically enforced in practice by syntactic properties of the milestone achieving entries. The first two are enforced as part of the operational semantics presented below.)

A artifact instance snapshot of type $R$ is an instance pre-snapshot $\sigma$ of type $R$ that satisfies the three GSM Invariants.

A artifact instance of $R$ is a sequence $\sigma_1, \ldots, \sigma_n$ of artifact instance snapshots of type $R$ such that $\sigma_1(\text{ID}) = \sigma_2(\text{ID}) = \cdots = \sigma_n(\text{ID})$. Intuitively, an instance of $R$ will correspond to a single conceptual entity of type $R$ that evolves as it moves through some business operations.

We now turn to GSM (pre-)snapshots and instances. A pre-snapshot of $\Gamma$ is an assignment $\Sigma$ that maps each type $R$ of $\Gamma$ to a set $\Sigma(R)$ of pre-snapshots of type $R$, and that satisfies the following structural properties:

- **Distinct ID’s:** If $\sigma$ and $\sigma'$ are distinct artifact instance snapshots occurring in the image of $\Sigma$, then $\sigma(\text{ID}) \neq \sigma'(\text{ID})$.

- **No dangling references:** If an artifact instance ID $\rho$ of type $\text{ID}_B$ occurs in the value of a non-ID attribute of some snapshot instance in $\Sigma(R')$ for some $R'$ in $\Gamma$, then there is an instance snapshot $\sigma$ in $\Sigma(R)$ such that $\sigma(\text{ID}) = \rho$.

Finally, a snapshot of $\Gamma$ is a pre-snapshot $\Sigma$ of $\Gamma$ such that each artifact instance pre-snapshot in the image of $\Sigma$ is an instance snapshot.
Let $\Gamma$ be a GSM model and $\Sigma$ a pre-snapshot of $\Gamma$. We now extend the function $\Sigma$ to ID’s and path expressions in the natural manner. In particular, if $\sigma \in \Sigma(R)$ is an instance pre-snapshot of type $R$ and $\sigma(ID) = \rho$, then we use $\Sigma(\rho)$ to denote $\sigma$, and for an attribute $A$ of type $R$ we use $\Sigma(\rho.A)$ to denote $\sigma(A)$. Let $R$ be an artifact type in $\Gamma$ with context variable $x$, and let $A$ be an attribute of $R$ with type $T$. Then the path expression $x.A$ has type $T$. If $T = ID_{R'}$ for some artifact type $R'$ in $\Gamma$, then the expression has type $ID_{R'}$, and can be extended to $x.A.A'$ for some attribute $A'$ in $R'$; this has the type of $A'$. This can be extended in the natural manner to paths of arbitrary length, where all attributes except the last have as type the ID of some artifact type in $\Gamma$. If $\tau(x) = x.<\text{path}>$ is such a path expression starting with $x$, then $\Sigma(\tau(x))$ is defined in the usual manner.

### 3.4 Events, Messages, Tasks, and Services

In the interest of simplifying notation, several assumptions are made in the formal GSM meta-model presented here, including the following.

1. A given 2-way service $F$ can be invoked by instances of exactly one artifact type, say $R$. The input and output signatures for a 2-way service are taken from the set $Att_{data}$ of data attributes of $R$. Intuitively, then, there is no indirection or mapping between the attributes of the information model and the input parameters for the service call when calling a 2-way service, and similarly when receiving the response from it. (In Barcelona, a binding is typically specified to enable such indirection.)

2. Messages sent by the environment are directed towards a single artifact instance, which is identified using the unique ID of that instance (intuitively, the system-generated ID of that instance). (It is straightforward to relax this restriction, by allowing associative specification of which artifact instance(s) are to be impacted by a message, and and allow a single message to impact multiple artifact instances.) Similar to services, the signature of the payload of a message sent by the environment to a target instance of type $R$ uses attribute names from the set of data attributes of $R$.

3. It is the responsibility of the ASC to determine, for an incoming event $e$, which artifact instances, if any, this event is applicable to (or in other words, what artifact instances are directly affected by the event). In the case of service call returns, either zero or one artifact instances will be identified, based on correlating the service name and time called. In the case of created instances, either zero or one artifact instances will be identified (because the payload of the service call may violate some condition). In the case where one artifact instance is identified, it is a newly created one. In the case of 1-way messages, zero, one, or many artifact instances may be identified. As discussed in Subsection 3.5 below, the ASC uses for incoming event type $E$ a parameterized query $Q_E[\ldots](\cdot)$, that ranges over the set of active artifact instances, and whose parameters are filled by the payload of the incoming event. Although not considered here, this query is typically derived from the sentries occurring in $\Gamma$.

4. Pre- and post-conditions may be added to the specifications of the services, following the spirit of semantic web services and [6, 14, 15]; this is not considered here.

5. Attribute assignment commands, e.g., $x.A_1 := x.A_2$ or $x.A_1 := x.A_2 + x.A_3$ are considered in this document to be services that are performed by the (external) environment. These can be modeled very precisely by environment services if pre- and post-conditions on services are incorporated. (Assignment commands are viewed as services in order to achieve a certain a separation of concerns, in particular, to separate the semantics of B-steps from the semantics of domain-specific operations. In
practice, assignment commands will be performed within an Artifact Service Center, but they will be treated similarly to external service calls.)

6. Timestamps for the events related to sending of messages, invoking services, etc., are not explicitly represented in the model. Rather, it is assumed that events from the environment arrive in a sequence, that they are processed by the artifact instances one at a time, and that for each event from the environment the artifact instances may generate zero or more events that are sent back to the environment before the next event is processed. Each B-step is assumed to be performed in a single instant of time, termed the logical timestamp of that B-step.

7. In the current document we do not explicitly consider messages from one artifact instance to another, nor instance creation requests from an artifact instance (to the ASC). However, both of these can easily be incorporated. In the case of messages, it is assumed that the message is sent to the ASC during one B-step; it is placed on the ASC’s queue of events; and it is processed in some subsequent B-step. In the case of artifact instance creation requests, similarly the service call is sent to the ASC during one B-step; it is placed on the ASC’s queue of events; it is processed in some subsequent B-step. That second B-step also has the effect of placing the service call return that holds the ID of the newly created artifact instance onto the ASC’s queue, and this is transmitted to the calling artifact instance during a third B-step.

We assume countably infinite sets MSG of message types (names), and SRV of (2-way) services (names), which are pairwise disjoint, and also disjoint from the sets TYPE_p, TYPE_ART, ATT, and ID_R for each type R \in TYPE_ART.

There are three kinds of message that can be generated by a ASC and sent to the environment, and three kinds of messages that can be generated by the environment and processed by the ASC. These are introduced now, and then described in a little more detail.

**Outgoing**: These are generated by the ASC and are received by the environment.

1-way message generated by artifact instance: We assume a family of outgoing message types. For an outgoing message type M, it will have an associated artifact type R, and a signature (A_1:Typ(A_1), ..., A_n:Typ(A_n)), where A_1, ..., A_n are data attributes for R and Typ is the type function for data attributes of R. A request to send a message of type M may be written as M(A_1:z_1, ..., A_n:z_n), where z_i is a terms that will evaluate to a constant of type Typ(A_i), i \in [1..n], and (grounded) occurrences of messages of type M may be written as M(A_1:c_1, ..., A_n:c_n), where the c_i’s are constants of the correct types.

2-way service call generated by artifact instance: We assume a family of (outgoing) 2-way service calls. For a 2-way service call type F, it will have an associated artifact type R, input signature (ID:ID_B, timeCalled:TIMESTAMP, A_1:Typ(A_1), ..., A_n:Typ(A_n)), and output signature (ID:ID_B, timeCalled:TIMESTAMP, A'_1:Typ(A'_1), ..., A'_m:Typ(A'_m)). The first two attributes of the input and output provide correlation information, so that the ASC knows which artifact instance should receive the service call return, and when the service was called. (The type of the service call return provides further correlation information for the artifact instance.) Analogous to 1-way messages, a request to invoke a service call of type F may be written as F^call(caller:z, timeCalled:z', A_1:z_1, ..., A_n:z_n) and a received service call return may be written as F^return(caller:z, timeCalled:z', A'_1:y_1, ..., A'_m:y_m) for appropriate terms z, z_i, y_j, and similarly for ground occurrences of these.
Service call return for Create artifact Instance: For each artifact type \( R \) there is a 2-way service call \( \text{create}^c_{\Gamma}(A_1: \text{Typ}(A_1), \ldots, A_n: \text{Typ}(A_n)) \) that the environment may invoke against the ASC, where each \( A_i \) is a data attribute of \( R \) and \( \text{Typ} \) is the data attribute typing function for \( R \) (see below). The service call return message for this service has form \( \text{create}^r_{\Gamma}(\text{ID:} \text{ID}_B, A_1: \text{Typ}(A_1), \ldots, A_n: \text{Typ}(A_n)) \). Here the first coordinate gives the ID of the newly created artifact instance and the remaining coordinates are copied from the service call, to provide correlation information to the calling entity. (If the first field is the null value, this indicates that the request failed, presumably because the input payload violated some condition.) Messages of this form will be generated by the ASC in response to incoming messages of type \( \text{create}^c_{\Gamma} \).

Incoming: These are generated by the environment and are received by the ASC.

1-way message from environment: We assume a family of incoming message types. For an incoming message type \( M \), it will have an associated artifact type \( R \), and analogous to the outgoing 1-way message types, it will have a signature \( (A_1: \text{Typ}(A_1), \ldots, A_n: \text{Typ}(A_n)) \), where \( A_1, \ldots, A_n \) are data attributes for \( R \) and \( \text{Typ} \) is the type function for data attributes of \( R \). Symbolic and ground occurrences of such messages may be written in a manner analogous to occurrences of outgoing 1-way messages.

Service call return from environment: As mentioned above, the environment may generate a message of form \( F^r_{\text{return}}(\text{caller:} \text{ID}_B, \text{timeCalled:} \text{TIMESTAMP}, A'_1: \text{Typ}(A'_1), \ldots, A'_m: \text{Typ}(A'_m)) \) where \( F \) is a 2-way service call associated with type \( R \).

Create artifact Instance service call: The environment may generate a message which has the form \( \text{create}^c_{\Gamma}(A_1: \text{Typ}(A_1), \ldots, A_n: \text{Typ}(A_n)) \), where \( R \) is an artifact type, \( A_i \) is a data attribute of \( R \) for \( i \in [1..n] \), and \( \text{Typ} \) is the typing function of \( R \) for data attributes. Intuitively, the intended semantics of such service calls is that the ASC will create a new instances of type \( R \), populate the attributes \( A_1, \ldots, A_n \) according to the payload of the service call, and then return to the caller the ID of the newly created instance. (There may be constraints on the payloads of create artifact instance calls; if violated the \( \text{ID} \) field of the service call return would be null.)

The set \( \text{IncEVENT} \) of incoming event types includes all possible incoming 1-way message types, 2-way service call return types, and the artifact instance creation 2-way service calls.

Definition: An event expression for an artifact type \( R \) with context variable \( x \) is an expression \( \xi(x) \) having one of the following forms.

- **Incoming event expression:** This includes:
  1. \( x.M \), where \( M \) is a incoming one-way message type targeted at instances of \( R \)
  2. \( x.F^r_{\text{return}} \), where \( F \) is a two-way service call type that can be invoked by instances of \( R \).
  3. \( x.\text{create}^c_{\Gamma} \), which can arise only in the case where \( x \) refers to a freshly created artifact instance, that resulted from the ASC creating an artifact instance in the course of executing an occurrence of the \( \text{create}^c_{\Gamma} \) service.

- **Internal event expression** (also known as status change event expression): This includes
  1. \( +\tau.m \) and \( -\tau.m \), where \( \tau \) is a well-formed path expression of form \( x.<\text{path}> \) with type \( \text{ID}_{\Gamma}^\tau \) for some artifact type \( R^\prime \) in \( \Gamma \), and where \( m \) is a milestone of type \( R^\prime.m \). Intuitively, an event occurrence of type \( +\tau.m \) arises whenever the milestone \( m \) of the instance identified by \( x.<\text{path}> \) changes value from false to true. The meaning of \( -\tau.m \) is defined analogously.
2. \( +\tau.active_S \) and \( -\tau.active_S \), where where \( \tau \) is a well-formed path expression of form \( x.<path> \) with type \( ID_{B'} \) for some artifact type \( R' \) in \( \Gamma \), and where \( S \) is a stage name of type \( R' \). Intuitively, an event occurrence of type \( +\tau.active_S \) arises whenever the stage \( S \) of the instance identified by \( x.<path> \) changes value from closed to open (inactive to active). The meaning of \( -\tau.active_S \) is defined analogously.

### 3.5 The Immediate Effect of an Incoming Event

Let \( \Sigma \) be a snapshot, \( e \) a grounded incoming event, and \( t \) a logical timestamp greater than all logical timestamps occurring in \( \Sigma \). Intuitively, the immediate effect of \( e \) on \( \Sigma \) at time \( t \), denoted \( \text{ImmEffect}(\Sigma,e,t) \), is the pre-snapshot that results from incorporating \( e \) into \( \Sigma \), including:

- Changing the values of the mostRecEventType and mostRecEventTime attributes of directly affected (or created) artifact instances, and
- Changing the values of data attributes of directly affected artifact instances (or initializing those data attributes in a newly created artifact instance), as indicated by the payload of \( e \).

The immediate effect does not incorporate any changes to status attributes, nor any firing of guard or milestone sentries. Note that in many cases, the immediate effect will be a pre-snapshot but not a snapshot.

A more formal definition of the immediate effect requires consideration of the three kinds of incoming event that can arrive into an ASC. Thus, service call returns, one-way incoming messages, and \( \text{create}^\text{call}_B \) messages, will be considered in turn.

To set the stage, suppose that \( e = E(A_1:c_1,\ldots,A_n:c_n) \) (or \( e = E(\text{caller}:\rho,\text{timeCalled}:t,A_1:c_1,\ldots,A_n:c_n) \) in the case of service call returns), where \( E \) is the event type of \( e \). We assume that there is a parameterized query \( Q_E[A_1:z_1,\ldots,A_n:z_n] \) (or \( Q_E[\text{caller}:z,\text{timeCalled}:z',A_1:z_1,\ldots,A_n:z_n] \)) that the ASC evaluates in order to determine which artifact instances, if any, are impacted by the incoming message. (In the case of a service call return, the \( \text{caller} \) attribute ensures this.) In the case of one-way messages there may be zero, one, or many impacted artifact instances. If \( E \) is of type \( \text{create}^\text{call}_B \), then \( Q_E \) has Boolean result; if it returns “true” then the ASC will first create a new artifact instance of type \( R \), and \( Q_E \) will return that instance.

Suppose now that \( E \) is a one-way message. Then \( \text{ImmEffect}(\Sigma,e,t) \) is the pre-snapshot \( \Sigma' \) obtained from \( \Sigma \) by modifying each artifact instance \( I \) in \( Q_E[A_1:z_1,\ldots,A_n:z_n] \) in the following way. First, set \( I.\text{mostRecEventType} := E \) and \( I.\text{mostRecEventTime} := t \). Second, for each data attribute \( A_j \), set \( I.A_j := c_j \). Note that if \( Q_E \) returns the empty set, then \( \text{ImmEffect}(\Sigma,e,t) = \Sigma \).

Suppose now that \( E \) is a service call return \( E^{\text{return}} \). If \( \rho \) (i.e., the ID of the artifact instance that called this occurrence of \( F \)) is not present in \( \Sigma \), then \( \text{ImmEffect}(\Sigma,e,t) = \Sigma \). Otherwise, let \( I \) be the artifact instance with ID \( \rho \). Then \( \text{ImmEffect}(\Sigma,e,t) \) is the pre-snapshot \( \Sigma' \) obtained from \( \Sigma \) by modifying each artifact instance \( I \) in \( Q_E[A_1:z_1,\ldots,A_n:z_n] \) in the following way. First, set \( I.\text{mostRecEventType} := E \) and \( I.\text{mostRecEventTime} := t \). Second, for each data attribute \( A_j \), set \( I.A_j := c_j \).

Finally, suppose that \( e \) is of type \( \text{create}^\text{call}_B \), that the parameterized query returns the answer “true”, and that \( \rho \) is the ID of the artifact instance of type \( R \) newly created by the ASC. In this case, \( \text{ImmEffect}(\Sigma,e,t) \) is the pre-snapshot \( \Sigma' \) obtained from \( \Sigma \) by including one new artifact instance of type \( R \) that has ID \( \rho \), where \( \rho.A_j = c_j \) for \( j \in [1..n] \), where the other data attributes are uninitialized and all status attributes are false.

For a snapshot \( \Sigma \), event \( e \), and timestamp \( t \), \( e \) has trivial effect on \( \Sigma \) at time \( t \) if \( \text{ImmEffect}(\Sigma,e,t) = \Sigma \), and has non-trivial effect otherwise.
3.6 GSM Business Steps (“B-steps”) and Logical Timestamps

We now describe the central notion of “GSM Business step” (“B-step”). Recall from Subsection 2.5 that an artifact model $\Gamma$ is used in connection with an abstract model of the (external) environment; this is typically denoted as $\Omega$. In application, an Artifact Service Center (ASC) is used as a container for the set of artifact instances of of the artifact types in $\Gamma$. The ASC can provide a variety of functionalities, including the relay of messages from an artifact instance out to the environment or to other artifact instances, and the relay of messages from the environment into the artifact instances.

The operational semantics for GSM are focused on the notion of B-steps, which correspond to the impact of a single incoming event occurrence $e$ at a logical timestamp $t$ on a snapshot $\Sigma$ of a GSM model $\Gamma$. This is illustrated in Figure 4. The semantics characterizes 5-tuples of the form $(\Sigma, e, t, \Sigma', Gen)$, where the following hold.

1. $\Sigma$ is the previous snapshot.
2. $e$ is a ground occurrence of an incoming event type associated with $\Gamma$.
3. $t$ is a logical timestamp which is greater than all logical time stamps occurring in $\Sigma$.
4. $\Sigma'$ is the next snapshot.
5. $Gen$ is the set of ground generated event occurrences, all of whose types are outgoing event types associated with $\Gamma$.

To illustrate the notion of B-step, we describe key aspects of the incremental formulation of the operational semantics. In this case, $\Sigma'$ is constructed in two phases (see Figure 5). The first is to incorporate $e$ into $\Sigma$, by computing $\text{ImmEffect}(\Sigma, e, t)$. (If $\text{ImmEffect}(\Sigma, e, t) = \Sigma$ then the incoming event $e$ is discarded and no B-step performed.) The second phase is to incorporate the effect of the guards, achieving sentries for milestones, invalidating sentries for milestones, and the first two GSM invariants. A family of ECA-like rules corresponding to these constructs is derived from $\Gamma$ (see Subsection 4.2). The second phase is achieved by building a sequence $\Sigma = \Sigma_0, \Sigma_1 = \text{ImmEffect}(\Sigma, e, t), \Sigma_2, \ldots, \Sigma_n = \Sigma'$ of pre-snapshots, where each step in the computation, called a micro-step, corresponds to the application of one ECA-like rule, and where no ECA-like rule can be applied to $\Sigma_n$. (There are restrictions on the ordering of rule application, as detailed in Subsection 5.1.) Here $\Sigma'$ corresponds to 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.

Although the creation of $\Sigma'$ and $Gen$ from $\Sigma$ and $e$ may take a non-empty interval of clock time, in the formal model we represent this as a single moment in time, called a logical timestamp. One can think of $t$ as the clock time at the moment when the system began processing event $e$. (There may be a queue of such events, so the start time of processing may be different than the time that the ASC receives $e$.)

![Figure 4: Illustration of a single GSM Business step (B-step)](image-url)
Each message in Gen is the result of opening an atomic stage with a message-generating task inside. The attributes of the message payloads are drawn from the data attributes of the the artifact instance, which remain fixed once ImmEffect(Σ, e, t) is computed. Thus, given the set of stages opened by a B-step it is straightforward to determine the set of messages that will be generated by that B-step. For this reason, the set Gen is not considered in the formalism below.

3.7 Sentries

Definition: A sentry for artifact type R is an expression χ(x) having one of the following forms, where x is the context variable of R:

- on ξ(x) if ϕ(x)
- on ξ(x)
- if ϕ(x)

where the following hold.

(a) If ξ(x) appears, then it is an event expression for R.

(b) If ϕ(x) appears, then ϕ(x) is a well-formed formula over the artifact types occurring in Γ that has exactly one free variable.

Expression ξ(x), if it occurs in the sentry, is called the (triggering) event. Expression ϕ(x), if it occurs in the sentry, is called the condition.

We now consider what it means for a pre-snapshot Σ to satisfy a sentry χ. Satisfaction of a condition by Σ is straightforward, and not considered further. Satisfaction of the triggering event, on the other hand, may involve not only Σ but also an incoming event e and also the logical timestamp t when this event is being processed.

Definition: Suppose R is an artifact type in GSM model Γ with context variable x, that Σ is a pre-snapshot of Γ, that e is a ground event occurrence of type E associated with R, and that t is a logical timestamp with the property that no logical timestamp greater than t occurs in Σ. Also, let ξ(x) occur in a sentry for R, and let ρ be the ID of some artifact instance pre-snapshot occurring in Σ(R). Then (Σ, e, t) satisfies ξ(x/ρ), denoted (Σ, e, t) |= ξ[x/ρ] if one of the following is true:
Suppose that $\xi(x)$ has the form $x.E$ where $E$ is an incoming event type associated with $R$. Then in $\Sigma$ it holds that $\rho._{\text{mostRecEventType}} = E$ and $\rho._{\text{mostRecEventTime}} = t$.

Suppose that $\xi(x)$ has the form $+\tau.m$ [respectively, $-\tau.m$], where $\tau$ has the form $x.<\text{path}>$ and evaluates to a value of type $\text{ID}_{B'}$, and $m$ is a milestone of $R'$. Then in $\Sigma$ it holds that $\rho._{\text{<path>}}.m = \text{true}$ [respectively, $\rho._{\text{<path>}}.m = \text{false}$] and $\rho._{\text{mostRecentUpdate}} = t$.

Suppose that $\xi(x)$ has the form $+\tau.active_S$ [respectively, $-\tau.active_S$], where $\tau$ has the form $x.<\text{path}>$ and evaluates to a value of type $\text{ID}_{B'}$, and $S$ is a stage of $R'$. Then in $\Sigma$ it holds that $\rho._{\text{<path>}}.active_S = \text{true}$ [respectively, $\rho._{\text{<path>}}.active_S = \text{false}$] and $\rho._{\text{<path>}}.active_S^{\text{mostRecentUpdate}} = t$.

Finally, if $\chi(x)$ is a sentry for type $R$, and $\rho, \Sigma, e, t$ are as above, then the notion of $(\Sigma, e, t)$ satisfies $\chi(\rho)$, denoted $(\Sigma, e, t) \models \chi(\rho)$, is defined in the natural manner.

## 4 PAC Rules and Polarized Dependency Graphs

This section introduces two pillars of the GSM operational semantics. First is a family of ECA-like rules, called “Prerequisite-Antecedent-Consequent (PAC)” rules (Subsection 4.2). Second is the notion of “Polarized Dependency Graph (PDG)” (Subsection 4.3), which is used to provide a form of stratification for the set of PAC rules associated with a GSM model.

### 4.1 Two intuitive principles

This subsection introduces and motivates two more-or-less equivalent intuitive “principles” that have guided the design of the GSM semantics. The first principle is phrased in terms of the incremental formulation of the GSM semantics, and the second is phrased in terms of the fixpoint formulation.

**Toggle-once Principle.** In a B-step $(\Sigma, e, t, \Sigma')$, if $\Sigma'$ is constructed from $(\Sigma, e, t)$ through the incremental application of PAC rules, then each status value attribute can change at most once during that construction.

**Change Dominates Principle:** In a B-step $(\Sigma, e, t, \Sigma')$, if the antecedents of two rules calling for opposite changes to a status value attribute $\rho.s$ of an artifact instance are both applicable to $\Sigma'$, then the rule that changes the value of $\Sigma(\rho.s)$ dominates over the other rule, and $\Sigma'(\rho.s) \neq \Sigma(\rho.s)$.

A primary intuitive motivation behind these principles is that a B-step is intended to be a “unit of business-relevant change”. In terms of the incremental semantics, this means that if a status value attribute changes during application of PAC rules, then that change should be visible (and incorporated into $\Sigma'$), rather than being hidden in the internal processing that computes $\Sigma'$. In terms of the Change Dominates Principle, this means that if there is a reason to change a status value attribute, then the change should be “documented” in one of the snapshots that is presented to the business, i.e., should be visible in between B-steps.

### 4.2 Prerequisite-Antecedent-Consequent Rules

All three formulations of the semantics for GSM are based on a variation of Event-Condition-Action (ECA) rules, called Prerequisite-Antecedent-Consequent rules, or PAC rules. Each such rule has three parts. The
rules can be interpreted in two ways. The first is in the context of the incremental formulation, at a point where we have built up the sequence $\Sigma = \Sigma_0, \text{ImmEffect}((\Sigma_0, e, t) = \Sigma_1, \ldots, \Sigma_i$. The other context is that of the fixpoint formulation, which focuses on the completed B-step $(\Sigma, e, t, \Sigma')$. We now give the intuition of the three components of the rules, in their grounded form, for both of these contexts.

**Prerequisite:** This part of the rule is considered relative to $\Sigma$ in both contexts. It may be thought of as a prerequisite for determining whether the rule is relevant to $(\Sigma, e, t)$.

**Antecedent:** This part of the rule is considered relative to $\Sigma_i$ in the incremental formulation, and relative to $\Sigma'$ in the fixpoint formulation. If the rule is relevant, then the antecedent can be thought of as the “if” part of a condition-action rule. As will be seen below, the antecedent will correspond to a sentry, and thus may include both a (first-order logic equivalent of a) triggering event and a condition.

**Consequent:** In the incremental formulation, if the rule is relevant, and if the antecedent is true in $\Sigma_i$, then the rule is considered to be eligible, and it may be fired to create $\Sigma_{i+1}$ according to the consequent. In the fixpoint formulation, if the rule is relevant and $\Sigma'$ satisfies the antecedent, then $\Sigma'$ should also satisfy the result called for by the consequent.

For the fixpoint formulation, the reader may wonder why the antecedent is considered relative to $\Sigma'$ rather than $\Sigma$. Intuitively, the focus is on creating $\Sigma'$ to be the fixpoint, in the spirit of logic programming,
of applying the PAC rules to \(\text{ImmEffect}(\Sigma, e, t)\). In logic programming, the fixpoint itself satisfies all of the if-then rules, considered as first-order logic formulas. Similarly, in GSM the fixpoint \(\Sigma'\) satisfies the AC part of each PAC rule, considered as a first-order logic formula.

Figure 6 describes two sets of abstract PAC rules that may be associated with a GSM model \(\Gamma\). Part (a) of the figure lists the templates for the set \(\Gamma_{\text{PACsimp}}\) of simplified PAC rules for \(\Gamma\). Part (b) lists the template \(\Gamma_{\text{PAC}}\) of (enhanced) PAC rules for \(\Gamma\) is the set of rules formed from \(\Gamma_{\text{PACsimp}}\) by removing rules generated from PAC-4\(^{\text{simp}}\), and adding all rules generated by PAC-4. Brief intuitions behind both sets of rules are now described.

Consider first the simplified PAC rules (Figure 6(a)). The first three kinds of rule are called explicit, and they correspond, respectively, to guards, to milestone achieving sentries, and to milestone invalidating sentries. The second three kinds of rule are called invariant preserving, because they focus on preserving the Invariants GSM-1 and GSM-2. (Recall that Invariant GSM-3 is assumed to be maintained by properties of the milestones themselves.)

Consider PAC-1. The antecedent is basically the guard that the rule is derived from. If \(S\) is the child of \(S'\), then the conjunct \(x.\text{active}_{S'}\) is added to the antecedent. The consequent corresponds to the intention of the guard to open stage \(S\). In the incremental semantics leading to the computation of \(\Sigma'\) from \((\Sigma, e, t)\), it is possible that both \(S'\) and \(S\) are closed in \(\Sigma\), that some incremental step opens \(S'\), and that a subsequent incremental step opens \(S\). In the final result \(\Sigma'\), both \(S\) and \(S'\) are open.

In general, the prerequisites are included to ensure that the Toggle-Once property is maintained.

We consider briefly PAC-4\(^{\text{simp}}\) and PAC-4, which are focused on Invariant GSM-1. PAC-4\(^{\text{simp}}\) follows the pattern and intuition of PAC-5 and PAC-6, and can be used in many situations. There are situations, however, in which it is desirable for a guard of a stage \(S\) to include as a condition that one or more of the milestones owned by \(S\) are currently not true. (This was illustrated in connection with stage Launching Line Items in Section 2.) In such cases, PAC-4\(^{\text{simp}}\) is needed so that the GSM model will still satisfy the well-formedness condition.

4.3 Stratification via Polarized Dependency Graphs

In the general case, the set of PAC rules of a GSM model \(\Gamma\) will involve a form of negation. As is well-known from logic programming and datalog, the presence of negation in rules can lead to non-intuitive outcomes. 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.

This subsection first presents an example that illustrates how non-intuitive outcomes can arise in the GSM context, then presents the “Polarized Dependency Graph (PDG)” of a GSM model, and uses that to specify a well-formedness condition on GSM models. (The requirement concerning the order of rule firing based on the PDG is described in Subsection 5.1 below.)

Consider Figure 7, and suppose that in a snapshot \(\Sigma\) we have for some artifact instance \(\rho\) that \(S1\) and \(S2\) are both open, that \(m1\) and \(m2\) are both false, that \(\rho.A = 20\), and that event \(e\) is to be processed and is applicable to \(\rho\). Suppose that the PAC rule templates are applied in the order suggested by the figure, that is: (1) milestone \(m1\) is achieved; (2) guard \(g3\) is triggered (since at this moment \(m1\) is true and \(m2\) is false); (3) stage \(S3\) is opened; (4) milestone \(m2\) is achieved; and (not numbered:) stages \(S1\) and \(S2\) are closed. Let \(\Sigma'\) be the result of these steps. Then \(\Sigma'\) is a snapshot. However, in \(\Sigma'\) we have that both \(m1\) and \(m2\) are true, and also the condition of guard \(g3\) is not true. Intuitively, there is no apparent reason, looking only at \(\Sigma\) and \(\Sigma'\) as to why \(S3\) became open in \(\Sigma'\). As defined in Subsection 5.2 below, we say that \(S3\) being open violates
the “inertial” property, which states that status attributes should not change from \( \Sigma \) to \( \Sigma' \) unless there is a visible.

Let \( \Gamma \) be a GSM model. We construct the polarized dependency graph (PDG) of \( \Gamma \), denoted PDG(\( \Gamma \)), as follows. The set \( V_\Gamma \) of nodes for PDG(\( \Gamma \)) contains the following for each artifact type \( R \) in \( \Gamma \):

- For each milestone \( m \) of \( R \), nodes \(+R.m\) and \(-R.m\)
- For each stage \( S \) of \( R \), nodes \(+R.active_S\) and \(-R.active_S\)
- For each guard \( g \) of \( R \), nodes \(+R.g\)

The set \( E_\Gamma \) of edges for PDG(\( \Gamma \)) is based largely on the rules in \( \Gamma_{PAC} \). In the following, \( R, R' \) range over not necessarily distinct artifact types in \( \Gamma \); \( s, s' \) range over not necessarily distinct status attributes of those types; and \( \odot, \odot' \) correspond to polarities, that is, they range over \{+, −\}. Also, let \( x, x' \) be the context variables for \( R, R' \), respectively.

- Suppose that \((\pi, \alpha, \gamma)\) is a PAC rule in \( \Gamma_{PAC} \) having the form of PAC-2, PAC-3, PAC-5, or PAC-6.
  - If \( \alpha \) includes as a triggering event the expression \( \odot' \tau(x).s' \) and \( \gamma \) is \( \odot x.s \), where \( \tau(x) \) evaluates to ID’s of type \( R' \), then include edge \((\odot' R'.s', \odot R.s)\).
  - If \( \alpha \) includes in its condition an expression \( \tau(x).s' \) and \( \gamma \) is \( \odot x.s \), where \( \tau(x) \) evaluates to ID’s of type \( R' \), then include edges \((+R'.s', \odot R.s)\) and \((-R'.s', \odot R.s)\).
• Suppose that \((\pi, \alpha, \gamma)\) is a PAC rule in \(\Gamma_{PAC}\) having the form of PAC-1, that is created because of guard \(g\) for stage \(S\) in type \(R\).
  
  - If \(\alpha\) includes as a triggering event the expression \(\circ'\tau(x).s'\), where \(\tau(x)\) evaluates to ID’s of type \(R'\), then include edge \((\circ' R'.s', +R.g)\).
  
  - If \(\alpha\) includes in its condition an expression \(\tau(x).s'\), where \(\tau(x)\) evaluates to ID’s of type \(R'\), then include edges \((+R'.s', +R.g)\) and \((-R'.s', +R.g)\).

• Suppose that \((\pi, \alpha, \gamma)\) is a PAC rule in \(\Gamma_{PAC}\) having the form of PAC-4, that is created because of guard \(g\) and milestone \(m\) for stage \(S\) in type \(R\).
  
  - If \(\alpha\) includes as a triggering event the expression \(\circ'\tau(x).s'\), where \(\tau(x)\) evaluates to ID’s of type \(R'\), then include edge \((\circ' R'.s', -R.m)\).
  
  - If \(\alpha\) includes in its condition an expression \(\tau(x).s'\), where \(\tau(x)\) evaluates to ID’s of type \(R'\), then include edges \((+R'.s', -R.m)\) and \((-R'.s', -R.m)\).

• Finally, if \(g\) is a guard for stage \(S\) in type \(R\), then include edge \((+R.g, +R.active_S)\).

This construction is illustrated in Figure 8.

**Definition:** A GSM model \(\Gamma\) is defined to be well-formed if \(PDG(\Gamma)\) is acyclic.

The acyclicity of the PDG is used to guide the ordering of rule application in the incremental formulation. For example, if in the PDG there is an edge from \(-R.m\) to \(+R'.g\), this indicates that in the incremental formulation, all rules that might make \(\rho.m\) false (for any \(\rho\) of type \(R\)) should be considered before any rule that might use \(\rho'.g\) to open its stage (for any \(\rho'\) of type \(R'\)). Lemma 5.1 below states that if such an ordering is followed, then the result is guaranteed to exist and be unique.

In some cases it is helpful to use a more lenient notion of well-formed, that is based on the acyclicity of all of the event-relativized PDGs. For an event type \(E\), the event-relativized PDG for \(\Gamma\) and \(E\) is constructed in the same manner as \(PDG(\Gamma)\), except that a rule \((\pi, \alpha, \gamma)\) is not considered if \(\pi\) is an incoming event type different from \(E\). (Although not considered here, the results of Section 5 hold for this more lenient notion of well-formed.)

## 5 Three Formulations of the GSM Operational Semantics

This section describes the three formulations of the GSM operational semantics, and then presents the equivalence theorem. The section closes with comments about B-steps considered in a series.

### 5.1 The Incremental Formulation

Assume that GSM model \(\Gamma\) is given, and let us focus on incorporating event \(e\) into snapshot \(\Sigma\) at time \(t\). Recall from Subsection 3.6 and Figure 5 that the incremental formulation is based on the construction of a sequence \(\Sigma = \Sigma_0, \Sigma_1 = ImmEffect(\Sigma, e, t), \Sigma_2, \ldots, \Sigma_n = \Sigma'\) (where \(\Sigma_1 \neq \Sigma\)).

Given \(\Sigma_j, j \geq 1\), an ground PAC rule \((\pi, \alpha, \gamma)\) is applicable to (or eligible to fire with) \(\Sigma_j\) if \(\Sigma \models \pi\) and \(\Sigma_j \models \alpha\). Applying (or firing) such a rule would yield a new pre-snapshot \(\Sigma_{j+1}\), that is constructed from \(\Sigma_j\) by “applying” the effect called for by \(\gamma\) (that is, toggling exactly one status attribute of one artifact instance).
In the incremental formulation, the application of the ground PAC rules must **comply** with the ordering implied by \(PDG(\Gamma)\), i.e., for each pair \(r, r'\) of ground rules with abstract actions \(\circ R.s \circ R'.s'\), respectively, if \(\circ R.s < \circ R'.s'\) then the rule \(r\) must be considered for firing before the rule \(r'\) is considered for firing.

**Lemma 5.1:** Suppose that \((\Sigma, e, t)\) is a snapshot, ground event, and time greater than all times in \(\Sigma\). Suppose further that \(\Sigma_1 = \text{ImmEffect}(\Sigma, e, t)\). Then there is at least one snapshot \(\Sigma'\) obtained by firing the rules of \(\Gamma_{PAC}\) in an ordering that complies with \(PDG(\Gamma)\). Furthermore, if \(\Sigma'\) and \(\Sigma''\) are constructed through applications of the rules in \(\Gamma_{PAC}\) using any rule firing order that complies with \(PDG(\Gamma)\), then \(\Sigma' = \Sigma''\).

**Proof (sketch):** The existence of at least one \(\Sigma\) follows primarily from the facts that a change called for by one rule cannot be “undone” be another rule (mainly due to the prerequisites of the rules and the definition of rule eligibility), and the fact that any sequence of rule firings will terminate (because there are only finitely many status attributes in a pre-snapshot). For uniqueness, assume that \(\Sigma'\) and \(\Sigma''\) are different end results, and let \(\circ \rho.s\) be a least ground status attribute that only one of \(\Sigma'\) or \(\Sigma''\) changes, where \(\rho\) is of type \(R\). Suppose without loss of generality that \(\Sigma'\) is the one where \(\rho.s\) changes. Since \(\Sigma', \Sigma''\) agree on all of the ground nodes that correspond to the abstract nodes preceding \(\circ \rho.s\) in \(PDG(\Gamma)\), the rule that triggered the change to \(\circ \rho.s\) in \(\Sigma'\) is also applicable in \(\Sigma''\), and could thus be fired there, yielding a contradiction. \(\square\)

**Definition:** A tuple \((\Sigma, e, t, \Sigma')\) **satisfies** the incremental formulation of the GSM operational semantics if \(\Sigma'\) is the unique result of applying the PAC rules in appropriate order to \(\text{ImmEffect}(\Sigma, e, t)\).

### 5.2 The Fixpoint Formulation

The fixpoint formulation for the GSM semantics is analogous to the fixpoint characterization used in logic programming. In our context, we start with \(\text{ImmEffect}(\Sigma, e, t)\) and characterize snapshots \(\Sigma'\) that satisfy two key mathematical properties stemming from \(\Gamma_{PAC}\).

Intuitively, the first property states that \(\Sigma'\) must comply with all of the demands of the PAC rules.

**Definition:** Given \(\Gamma\) and \((\Sigma, e, t)\) as above, with non-trivial immediate effect, then snapshot \(\Sigma'\) is **compliant** with respect to \(\Gamma_{PAC}\) and \((\Sigma, e, t)\) if

1. \(\Sigma'\) and \(\text{ImmEffect}(\Sigma, e, t)\) agree on all data attributes, and
2. for each ground PAC rule \((\pi, \alpha, \gamma)\) of \(\Gamma_{PAC}\), if \(\Sigma \models \pi\) and \(\Sigma' \models \alpha\), then \(\Sigma' \models \gamma\).

Intuitively, the second property states that if a status attribute toggles between \(\Sigma\) and \(\Sigma'\), then that toggling must be “justified” by some ground PAC rule.

**Definition:** Given \(\Gamma\) and \((\Sigma, e, t)\) as above, with non-trivial immediate effect, then snapshot \(\Sigma'\) is **inertial** with respect to \(\Gamma_{PAC}\) and \((\Sigma, e, t)\) if the following holds for each artifact instance ID \(\rho\) in \(\Sigma_1 = \text{ImmEffect}(\Sigma, e, t)\) having type \(R\), and each status attribute \(s\) of type \(R\): if \(\Sigma_1(\rho.s) \neq \Sigma'(\rho.s)\) then there is some ground PAC rule \((\pi, \alpha, \gamma)\) of \(\Gamma_{PAC}\) such that: (a) \(\Sigma_1 \models \pi\); (b) \(\Sigma' \models \alpha\); and (c) the value of \(\Sigma'(\rho.s)\) corresponds to the application of \(\gamma\).

**Definition:** A tuple \((\Sigma, e, t, \Sigma')\) **satisfies** the fixpoint formulation if \(\Sigma'\) is compliant and inertial with respect to \(\Gamma\) and \((\Sigma, e, t)\).
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 an extended FOL formula. The construction of the overall formula is reminiscent of constructions used for logic programming with negation, and in particular, when characterizing "negation as failure" [22].

The formula will work on structures of the form $(\Sigma, e, t, \Sigma')$. To express the formula over this structure, following the convention from verification theory, we use atomic formulas of the form $\phi(x_1, ..., x_n)$ to range over $\Sigma$, and of form $\phi'(x'_1, ..., x'_n)$ to range over $\Sigma'$. Also, given a formula $\alpha$ involving un-primed variables, we use $\alpha'$ to denote the formula obtained from $\alpha$ by priming all of the variables (and thus making all of the atomic formulas relevant to $\Sigma'$).

For a type $R$ of $\Gamma$, status attribute $s$ in $R$, and polarization $\odot$, let $\text{Cnsq}(\odot R.s)$ be the set of rules in $\Gamma_{PAC}$ whose consequent is $\odot R.s$. Also, define $\psi_{+R.s}$ to be

$\left( (-R.s \wedge \bigvee_{(\pi,\alpha,+R.s)\in\text{Cnsq}(+R.s)} (\pi \wedge \alpha') \rightarrow R.s') \wedge \right)$

$\left( (-R.s \wedge \bigwedge_{(\pi,\alpha,+R.s)\in\text{Cnsq}(+R.s)} \neg(\pi \wedge \alpha') \rightarrow \neg R.s' \right)$

and define $\psi_{-R.s}$ to be

$\left( (R.s \wedge \bigvee_{(\pi,\alpha,-R.s)\in\text{Cnsq}(-R.s)} (\pi \wedge \alpha') \rightarrow \neg R.s' \right) \wedge$

$\left( (R.s \wedge \bigwedge_{(\pi,\alpha,-R.s)\in\text{Cnsq}(-R.s)} \neg(\pi \wedge \alpha') \rightarrow R.s' \right)$

Finally, the closed-form formula $\Psi_{\Gamma}$ is defined as the conjunction of all of the formulas $\psi_{\odot R.s}$, along with a formula $\psi_{\text{incorp-event}}$ (not defined here) that states that the data attributes of $\Sigma'$ match those of $\text{ImmEffect}(\Sigma, e, t)$ (and that a new artifact instance has been created if $e$ calls for that to happen).

**Definition:** A structure $(\Sigma, e, t, \Sigma')$ satisfies the closed-form formulation of the GSM operational semantics if $(\Sigma, e, t, \Sigma') \models \Psi_{\text{Gamma}}$.

5.4 The Equivalence Theorem

The equivalence of the three formulations of the GSM semantics holds for all GSM models $\Gamma$ such that $\text{PDG}(\Gamma)$ is acyclic.

**Theorem 5.2:** Let $\Gamma$ be a well-formed GSM model; $\Sigma, \Sigma'$ two snapshots of $\Gamma$, $e$ a ground incoming event, $t$ a timestamp that is after all timestamps in $\Sigma$. Assume $\text{ImmEffect}(\Sigma, e, t) \neq \Sigma$. Then the following are equivalent.

- $(\Sigma, e, t, \Sigma')$ satisfies the incremental formulation.
- $(\Sigma, e, t, \Sigma')$ satisfies the fixpoint formulation.
- $(\Sigma, e, t, \Sigma')$ satisfies the closed-form formulation.

There is exactly one $\Sigma'$ that satisfies these properties.

**Proof (sketch):** The second two formulations are equivalent because $\Psi_{\Gamma}$ captures in extended FOL precisely the conditions of compliant and inertial. Let $\Sigma'$ be constructed according to the incremental formulation. Note that the application of rules is monotonic, in the sense that in the sequence of rule firings, each
rule applied makes a new change to the preceding pre-snapshot, and no change is “undone”. Also, if a rule is fired, then all attributes in its antecedant cannot change after that rule firing. Finally, since no rule can be applied to Σ', we have the compliance property. For inertial, note that a status attribute is changed from Σ to Σ' only if there is a rule firing that changed it. For the opposite direction, given Σ' that is inertial and compliant, one can identify a ground rule that justifies each change between Σ and Σ'. Order these rules according to PDG(Γ). Based on this, create a sequence of pre-snapshots that satisfies the incremental formulation. Uniqueness follows from Lemma 5.1. □

As a corollary, that follows from Lemma 5.1 and the equivalence theorem, we have:

**Corollary 5.3:** Let Γ and (Σ, e, t) be as in the previous theorem, where Γ is well-formed and ImmEffect(Σ, e, t) ≠ Σ. There there is exactly one result of (Σ, e, t) in Γ.

5.5 B-steps in series

This subsection briefly considers situations in which it makes intuitive sense to consider a cluster of B-steps as a single unit. Recall that 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 b₁ and is closed in some subsequent B-step b₂. Because the assignment is purely computational, it makes sense to have b₂ happen immediately after b₁. The same is true if B-step b₁ generates a message to the ASC intended for another artifact instance, or that calls for creation of an artifact instance, and b₂ processes that message. In practice, we define an mega-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 automated actions within the BSC. Mega-B-steps are not guaranteed to terminate, nor to be unique.

We also note that there are some unnatural corner cases, such that once a B-step has been computed there may be a rule r that is applicable. This can arise if the action of r “undoes” a change made during the preceding B-step. In this situation, we say that the GSM model is not stable. In such cases, it may be appropriate to include a B-step for applying such rules r into the mega-B-step.

One way this might arise is illustrated in Figure 9. Assume for this example that S₁ and S₂ are open for some artifact instance ρ, and that ρ.e arrives where e has type E. Then m₁ can be achieved, S₁ can close, milestone m₂ can be achieved, and S₂ can close. At this point, it may seem appropriate to open S₁, given that the guard g₁ is true. However, the prerequisite of this rule, which states that stage S₁ is open, is not satisfied at this point.

In general, designers should be advised not to create non-stable GSM models. It remains open to develop an efficient test for non-stability of a model.
6 Related Work

The GSM approach draws on previous work on ECA systems (e.g., [24]), but develops a specialized variant useful in the context of data-centric management of business operations and processes.

There is a strong relationship between the artifact paradigm and Case Management [32, 13, 33]. 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) instances 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” [8], to manage “distributed enterprise services” [5], and to provide cross-silo visibility into the management of engineering changes in large-scale manufacturing. GSM is a natural evolution from the earlier artifact meta-models [26, 9, 27], in which the lifecycle models are based on variants of finite state machines, to a more declarative style of lifecycles that are based primarily on conditions. Somewhat analogously, Adaptive Case Management [28] is a natural evolution from earlier case management approaches, that permits more freedom 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 is focused on the development of a precise mathematical definition for the operational semantics, and may offer a natural vehicle for providing a precise operational semantics for Advanced Case Management systems.

The AXML Artifact model [2, 23] supports a declarative form of artifacts using Active XML [1] as a basis. The approach takes advantage of the hierarchical nature of the XML data representation used in Active XML. In contrast, GSM uses milestones and hierarchical stages that are guided by business considerations.

DecSerFlow [31] 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 procedural system that permits the use of a rich rules-based paradigm for determining, at any moment in time, what activities should be performed next.

There is a loose correspondence between the artifact approach and proclets [30]. 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. The artifact approach places more emphasis than the proclets approach on the data that is held and maintained by each component. Artifact lifecycles routinely use the data in conditionals that govern internal behavior, reaction to incoming events, and interactions between components. The information model and current data held by an artifact instance can be exposed to other artifact instances (and the external environment), subject to access controls. Furthermore, GSM supports the use of conditions testing the data of other artifact instances and triggering based on status changes in other artifact instances to help govern interactions between components. In contrast, 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.

7 Conclusions

This paper provides an important extension to on-going work in the general area of event-driven, declarative, and data-centric business process management, using the Business Artifacts paradigm. Building on two previous papers that introduce the Guard-Stage-Milestone (GSM) approach for specifying business
entity lifecycles, this paper describes how GSM supports the interaction between artifact instances, both through triggering of (status change) events and through conditions that directly test the data in other artifact instances. The overall behavior of a GSM system is specified in terms of ECA-like rules, providing a declarative flavor. A precise operational semantics is given, and three formulations for that semantics are shown to be equivalent.

We mention a few research directions now being pursued. One is to better understand the behavior of sequences of B-steps, as indicated in Subsection 5.5. It will be important to add constructs to maintain transactional consistency between stages that are running in parallel. We are currently exploring the use of read- and write-locks, that are set or released as part of B-steps, in order to ensure that parallel executions of stages do not conflict in undesired ways. The incorporation of such locks will require some extensions to the techniques used to obtain Theorem 5.2.

In terms of expressive power of the underlying constructs, we are exploring the addition of a “map” operator, to be used in connection with a single stage, so that multiple occurrences of the stage can run simultaneously, one for each member of a collection attribute (that satisfies conditions specified by the guard). Also, we hope to enable from within a GSM instance that there can be a atomic stage that directly involves two or more artifact instances. Another extension would be to permit triggering events in sentries that are “complex”, in the sense that they might refer to more than one atomic event occurrence.

Another very important direction is to add an explicit notion of performers – either human or automated – that can execute both the tasks in atomic stages and can guide the overall operation of artifact instances through the sending of 1-way messages and artifact instance creation requests. In fact, it appears that the free-form, relatively ad hoc, event-driven nature of GSM stages can explicitly support a rich style of interaction between a human and a portion of business operations. The team is exploring the idea of a “delegation hierarchy”, that follows the pattern of the stage hierarchy, in which a composite stage is the responsibility of some person $P_1$, and she can assign responsibility (perhaps to herself, perhaps to others) for each substage, and those people can in turn either perform the substage in its entirety or delegate further.

Acknowledgements

The authors thank many people in the extended Project ArtiFact team, the ACSI team, and others who have been involved with GSM either in the design or in its initial uses. Most notable among these are: Diego Calvenese, David Cohn, Giuseppe De Giacomo, Nirmit Desai, Alin Deutsch, Amit Fisher, Nanjangud C. Narendra, Jianwen Su, John Vergo, and Victor Vianu.

References


