Business Artifacts with Guard-Stage-Milestone Lifecycles: Managing Artifact Interactions with Conditions and Events

Richard Hull*[1], Elio Damaggio†[2], Riccardo De Masellis‡[3], Fabiana Fournier‡[4],
Manmohan Gupta[5], Fenno (Terry) Heath III[1], Stacy Hobson[1], Mark Linehan[1],
Sridhar Maradugu§[6], Anil Nigam[1], Piwadee (Noi) Sukaviriya[1], Roman Vaculín[1]

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

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 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. 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 studied in most previous work, and support hierarchy and parallelism within a single artifact instance. This paper presents the formal operational semantics of GSM, with an emphasis on how interaction between artifact instances is supported. Such interactions are supported both through testing of conditions against the artifact instances, and through events stemming from changes in artifact instances. Building on a previous result for the single artifact instance case, a key result here shows the equivalence of three different formulations of the GSM semantics for artifact instance interaction. One formulation is based on incremental application of ECA-like rules, one is based on two mathematical properties, and one is based on the use of first-order logic formulas.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; H.4 [**Information Systems Applications**]: Office Automation—*Workflow management*

General Terms

Design, Theory, Verification

Keywords

Business Artifact, Business Entity with Lifecycle, Business Operations Management, Business Process Management, Case Management, Data-centric Workflow, Declarative Workflow, Event-Condition-Action Systems

1. INTRODUCTION

There is increasing interest in frameworks for specifying and deploying business operations and processes that combine both data and process as first-class citizens. One such approach is called Business Artifacts, or "Business Entities (with Lifecycles)"; this has been studied by a team at IBM Resarch for several years [18, 5, 17], and forms one of the underpinnings for the EU-funded Artifact-Centric Services Interoperation (ACSI) project [2]. Business 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 [12, 7] introduced a declarative approach to specifying artifact lifecycles, using the *Guard-Stage-Milestone* (GSM) lifecycle meta-model.[1] The current paper describes research on the formal specification and properties of an operational semantics for GSM, with an emphasis on (a) how GSM supports declarative specification of interaction

*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.

DEBS'11, July 11–15, 2011, New York, New York, USA.
Copyright 2011 ACM 978-1-4503-0423-8/11/07 ...$10.00.

[1] 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.

between business artifacts, (b) how ECA-like rules are used to provide one of the three equivalent formulations of the GSM operational semantics, and (c) how results of [7] are generalized to the context of multiple artifact types and instances.

As described in [12], a core motivation of the research leading to GSM has been to create a meta-model for specifying business operations and processes that is based on intuitively natural constructs that correspond closely to how business-level stakeholders think about their business. GSM supports high-level and more details views of the operations, and supports a spectrum of styles for specifying the operations, from the highly "prescriptive" (as found in, e.g., BPMN) to the highly "descriptive" (as found in Adaptive Case Management systems). Importantly, GSM provides a natural, modular structuring for specifying the overall behavior and constraints of a model of business operations in terms of ECA-like rules.

There are four key elements in the GSM meta-model: (a) *Information Model* for artifacts, as in all variations of the artifact paradigm; (b) *Milestones*, which correspond to business-relevant operational objectives and are achieved (and possibly invalidated) based on triggering events and/or conditions over the information models of active artifact instances; (c) *Stages*, which correspond to clusters of activity intended to achieve milestones; and (d) *Guards*, which control when stages are activated. Both milestones and guards are controlled in a declarative manner, based on triggering events and/or conditions.

This paper is focused primarily on the *operational semantics* used by the GSM 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 an (extended) first-order logic formula. This permits the application of previously developed verification techniques to the GSM context. (The previous work, [3, 9, 6], assumed that services were performed in sequence, while in GSM services and other aspects may be running in parallel.)

This paper describes and motivates these three formulations of the semantics, and sketches the proof of their equivalence. Reference [7] provides a more rigorous presentation of these results, using an abstract version of GSM that permits a focus on the essential aspects of the meta-model. The current paper expands on those results in two specific ways: (1) Generalize from a single artifact type and single artifact instance to a context of multiple artifact types and instances, including the possibility of new artifact instance creation in B-steps; and (2) Providing support for artifact instance interaction based on multi-instance conditions, and status-change events in one instance directly impacting another instance.

Although not a focus of the current paper, 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; this follows a central philosphy of artifacts, namely, that the attribute data is business relevant and should be exposed (subject to appropriate access restrictions).

The GSM meta-model describe here is being implemented in the Barcelona prototype at IBM, a descendant of the Siena prototype [4]. Due to space limitations, the exposition here is brief with a focus on the intuitions, main concepts, and main results; more details are available in [13].

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. Section 3 introduces some of the formalism used to specify GSM models. Section 4 introduces the ECA-like "Prerequisite-Antecedent-Consequent (PAC)" rules, and the "Polarized Dependency Graph (PDG)" which is used to define a well-formedness condition on GSM models, and to guide the application of the PAC rules. Section 5 presents the three formulations of the GSM operational semantics and their equivalence. Section 6 describes related work, and Section 7 offers brief conclusions.

2. MOTIVATING EXAMPLE

This section provides an informal introduction to GSM by describing how it can be used to model a Requisition and Procurement Orders (RPO) scenario.[2]

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.

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 parts 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 a Customer to the Manufacturer), 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 instance manages a single procurement order (from the Manufacturer to a Supplier), from when

[2] The authors thank Joachim (Jim) Frank of IBM for first introducing them to a form of this problem scenario. We are using a simplified version of the scenario here.

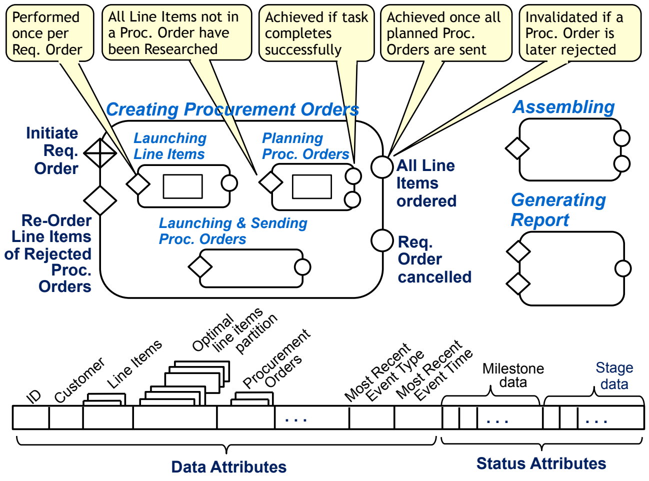

Performed once per Req. Order
All Line Items not in a Proc. Order have been Researched
Achieved if task completes successfully
Achieved once all planned Proc. Orders are sent
Invalidated if a Proc. Order is later rejected
Creating Procurement Orders
Assembling
Initiate Req. Order
Launching Line Items
Planning Proc. Orders
All Line Items ordered
Generating Report
Re-Order Line Items of Rejected Proc. Orders
Launching & Sending Proc. Orders
Req. Order cancelled
ID
Customer
Line Items
Optimal line items partition
Procurement Orders
Most Recent Event Type
Most Recent Event Time
Milestone data
Stage data
. . .
Data Attributes
Status Attributes

Figure 1: Sketch of artifact type for Requisition Order

it is initially sent to a supplier to the receipt of the goods or rejection by the supplier.

Due to space limitations, 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 Surrounding Framework for GSM Systems

In the general setting, an *Artifact Service Center* (*ASC*) is used to maintain a family of related artifact types and their associated instances. Speaking intuitively, 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. In particular, values of the data attributes are in general provided by human or automated agents that perform the tasks in a GSM model.

As noted in the Introduction, 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 as it moves through the business operations. In practice, this view might be materialized, virtual, or a hybrid.

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. Each milestone corresponds to one alternative way that the stage might reach completion. A stage becomes "inactive" (or "closed") when one of its milestones is achieved. Intuitively, this is because the overall motivation for executing a stage is to achieve one of its milestones.

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

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 acheived or invalidated. The triggering events may be incoming or internal to the ASC, and both the internal events and the conditions may refer to the artifact instance under consideration, and to other artifact instances in the ASC.

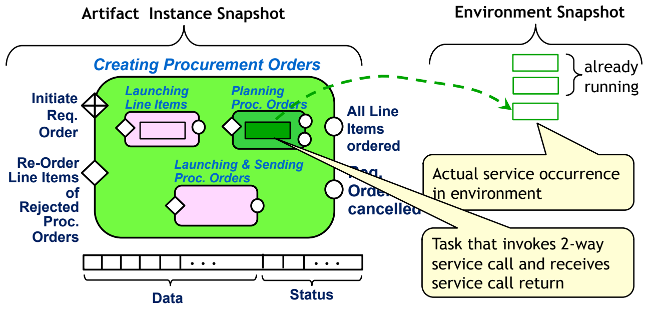
2.3 Drill-down into the RO Artifact Type

Figure 1 illustrates the key components of the GSM meta-model through a sketch of the Requisition Order artifact type. This artifact type is 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 very loosely, these attributes are generally filled up from left to right, although they may be overwritten. The *Status Attributes* are also illustrated; these hold information about the current status and update time of all milestones (true or false) and all stages (open or closed).

The upper portion of Figure 1 illustrates parts of the *lifecycle model* of the RO artifact type. *Milestones* are shown as small circles associated with stages. Some of the sentries for the milestones are suggested in the call-out boxes of that figure. For example, one of the milestone achieving sentries of All Line Items ordered will become true if all of the planned PO's have been sent. 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 be *invalidated* or "compromised", and become false. As an example of invalidation, 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*. (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*, which in this case call services that exist outside of the ASC. involve activities that are modeled outside of the GSM model. There are three categories of task: to (a) invoke 2-way service call against the "environment", (b) send 1-way message to the environment (or to another artifact instance), and (c) send a 1-way message to the ASC calling for the creation of a new instance.

The diamond nodes are *guards*. If a guard for a currently closed stage S becomes true, and if there is a parent of S which is already open, then S becomes open. The guards shown in Figure 1 are labeled with names as a convenience; in the formal model these names are not accessible to the sentries. The diamonds with a cross are "bootstrapping" guards, which are used to indicate the conditions under which new artifact instances may be created.



Actual service occurrence in environment

Task that invokes 2-way service call and receives service call return

Figure 2: Invoking 2-way service calls that run in the environment

Two broad categories of events may be used in sentries: "incoming" and "internal". One form of *incoming event* corresponds to events that come from the environment *into* one or more artifact instances. There are three kinds of such event: (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. The second form of *incoming event* arises when one artifact instance generates a message intended for another artifact instance (or to create an artifact instance); in this case the ASC acts as an intermediary. Such events, when passed by the ASC to the target (or newly created) artifact instances, behave in much the same way as the first form of incoming event. There are three categories of artifact to artifact incoming events: (d) 1-way messages; (e) a request to create a new artifact instance, and (f) a return message to an artifact instance creation request, that holds the ID of the newly created artifact instance.

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).

2.4 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 (dark) green if it is currently open, and in (light) 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). This service may eventually 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 (e.g., if a manager determines that the overall activity should be stopped). The service occurrence might continue to run, but the service call return would be ignored by the artifact instance that called it. In practical situations, the agent agent performing the service occurrence may be alerted that the service occurrence should be aborted. Also, in a practical setting, a richer style of interaction between an agent performing a service and the ASC might be supported.

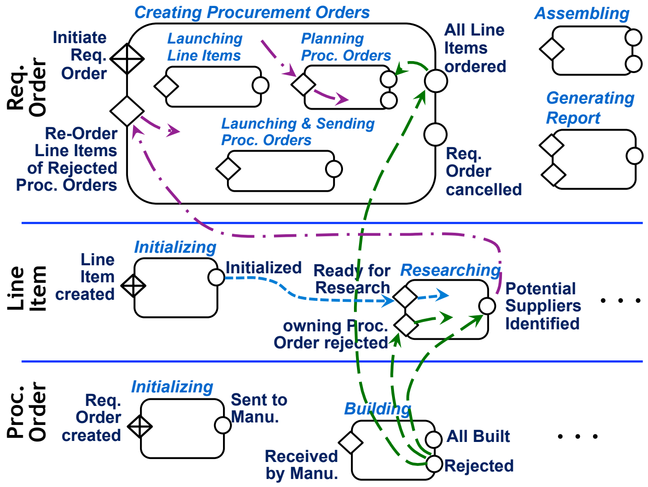


Figure 3: Interactions between artifact types

2.5 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.

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.

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 notation used in the current paper, this is written as +l.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 Initialized is achieved, the guard 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) coming from the PO milestone Rejected. These correspond to kinds of actions that might occur in a B-step if some PO instance p achieves the milestone Rejected. In particular, the milestone 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 All Line Items ordered is invalidated. Turning to the impact on LI, for each LI instance l that is "owned" by p, the milestone Potential Suppliers identified for l, if true, is invalidated. Also, the guard owning Proc. Order rejected of the Researching stage is triggered, and this stage is re-opened.

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 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.

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 *true*. However, the intention is that the stage Launching Line Items should occur just once, and not repeatedly. This can be achieved here by using not p.m as the guard g. This device, of including the negations of milestones of a stage as conjuncts into a guard, is a useful pattern when using guards without triggering conditions.

2.6 Adding Flowchart Arrows to Lifecycle Models

We briefly mention that conditional flowchart arrows can be added as a formal part of a GSM lifecycle model. The specification of the most common kind of flowchart arrow arises when there is a guard g of form "**On** $+x.m$ **if** φ" where m is a milestone. This is essentially equivalent to having a flow arrow from the milestone m to guard g, where the arrow is annotated with the condition φ, meaning that the arrow should be traversed when the miletone is achieved but only if φ is true at that time. If desired, additional kinds of arrow can be incorporated, corresponding to different combinations of how one kind of status change (i.e., achieving or invalidating a milestone, opening or closing a stage) leads to another.

3. GSM META-MODEL

This section formally introduces the GSM meta-model. As such, there will be some redundancy with the informal introduction to GSM presented in Section 2. (Full definitions are provided in [13].)

3.1 Domain Types and Extended FOL

The artifact meta-model presented here supports the use of arbitrary scalar types, including Boolean, integer, etc. Two specialized types are **IncEVENT**, which ranges over the possible (names of) types of incoming events, and **TIMESTAMP**, which ranges over the "logical" timestamps corresponding to the times that B-steps are executed. Also, for each artifact type R we assume a set $\mathbf{ID}_R$ of ID's for instances of type R.

The domains of all of these types are extended with the null value $\bot$. Collections of scalars, and collections of records of scalars are supported. Finally, the family of all *permitted* types for artifact attributes is denoted $\mathbf{TYPES}_{\text{permitted}}$.

(In the practical GSM meta-model, arbitrarily deep nesting of the relation construct is permitted.)

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) the use of ordered pairs to represent values, where one coordinate holds a Boolean that indicates whether the value is $\bot$ or not; (iv) the "dot" notation to form path expressions (both into record types and to follow links based on artifact IDs); (v) a binary predicate $\in$ to test membership in a collection; and (vi) quantification over both collection types in artifact instances and over the full domain of currently active instances of an artifact type. It is well-known that expressions in this extended FOL can be transformed into equivalent expressions in classical FOL.

3.2 Artifact Types and GSM Models

This subsection introduces the notion of artifact type, which provides the structure for instances of business artifacts, and the notion of GSM model, which provides the structure for families of related business artifact types and their instances.

Definition: An *artifact type* has the form

$$(R, x, \mathit{Att}, \mathit{Typ}, \mathit{Stg}, \mathit{Mst}, \mathit{Lcyc})$$

where the following hold:

- R is the *name* of the artifact 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 *Lcyc*.
- *Att* is the set of attributes of this type. *Att* is partitioned into the set Att_{data} of *data attributes* and Att_{status} of *status attributes* (see below).
- *Typ* is the *type function* for the data attributes, i.e., $\mathit{Typ}{:}\mathit{Att} \rightarrow \mathbf{TYPES}_{\text{permitted}}$.
- *Stg* is the set of *stage names*, or simply, *stages*.
- *Mst* is the set of *milestone names*, or simply, *milestones*.
- *Lcyc* is the *lifecycle model* of this artifact type (defined below).

The set Att_{data} must include an attribute *ID*, which holds the identifier of the artifact instance.

Additional restrictions are made so that events and their time of occurrence can be tested by sentries. In particular, Att_{data} should include two attributes: *mostRecEventType*, which holds the type of the most recent incoming event that affected this artifact instance; and *mostRecEventTime*, which holds the logical timestamp of the processing of that most recent incoming event. For each milestone $m \in M$, there are two attributes in Att_{status}, namely: a Boolean *milestone status value* attribute, denoted simply as m, and a *milestone toggle time attribute*, denoted as $m^{\mathit{mostRecentUpdate}}$, which holds the most recent time that the status value changed. Analogously, for each stage $S \in \mathit{Stg}$, there are two attributes in Att_{status}: a Boolean *stage status value* attribute, denoted as active_S, and a *stage toggle time attribute*, denoted as $\mathit{active}_S^{\mathit{mostRecentUpdate}}$.

An artifact type $(R, x, \mathit{Att}, \mathit{Typ}, \mathit{Stg}, \mathit{Mst}, \mathit{Lcyc})$ is often referred to using simply its name R. We use $\mathbf{ID}_R$ to denote the type of IDs of artifact instances of R.

The structure of artifact type lifecycle models is defined next.

Definition: Let $(R, x, \mathit{Att}, \mathit{Typ}, \mathit{Stg}, \mathit{Mst}, \mathit{Lcyc})$ be an artifact type. The lifecycle model *Lcyc* of R has structure

$$(\mathit{Substages}, \mathit{Task}, \mathit{Owns}, \mathit{Guards}, \mathit{Ach}, \mathit{Inv})$$

and satisfies the following properties.

- *Substages* is a function from *Stg* to finite subsets of *Stg*, where the relation $\{(S, S') \mid S' \in \mathit{Substages}(S)\}$ 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 *Stg* to *tasks* (defined in Subsection 3.4).
- *Owns* is a function from *Stg* to finite, non-empty subsets of *Mst*, such that $\mathit{Owns}(S) \cap \mathit{Owns}(S') = \emptyset$ for $S \neq S'$. A stage S *owns* a milestone m if $m \in \mathit{Owns}(S)$.
- *Guards* is a function from *Stg* to finite, non-empty sets of *sentries* (defined in Subsection 3.7). For $S \in \mathit{Stg}$, an element of $\mathit{Guards}(S)$ is called a *guard* for S.
- *Ach* is a function from *Mst* to finite, non-empty sets of sentries. For milestone m, each element of $\mathit{Ach}(m)$ is called an *achieving sentry* of m.
- *Inv* is a function from *Mst* to finite sets of sentries. For milestone m, each element of $\mathit{Inv}(m)$ is called an *invalidating sentry* of m.

If $S \in \mathit{Substages}(S')$, 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 Γ of artifact types with form $(R_i, x_i, \mathit{Att}_i, \mathit{Typ}_i, \mathit{Stg}_i, \mathit{Mst}_i, \mathit{Lcyc}_i)$, $i \in [1..n]$, that satisfies the following:

- **Distinct type names:** The artifact type names R_i are pairwise distinct.
- **No dangling type references:** If an artifact type $\mathbf{ID}_B$ is used in the artifact type R_i for some $i \in [1..n]$, then $R = R_j$ 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 Γ be as above. As will be seen, the sentries of guards and milestones in one GSM type R_i of Γ may refer to the values of attributes in Att_j for type R_j for any $j \in [1..n]$, not just for $j = i$.

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". We shall use pre-snapshots to describe the incremental construction of a new GSM snapshot in a B-step.

Let Γ be a GSM model, and $(R, x, \mathit{Att}, \mathit{Typ}, \mathit{Stg}, \mathit{Mst}, \mathit{Lcyc})$ be an artifact type in Γ. In this context, an artifact instance *pre-snapshot* of type R is an assignment σ from *Att* to values, such that for each $A \in \mathit{Att}$, $\sigma(A)$ has type $\mathit{Typ}(A)$. (Note that $\sigma(A)$ may be $\perp$ except for when $A = \mathit{ID}$.)

Let σ be an artifact instance pre-snapshot and let $\rho = \sigma(\mathit{ID})$. If understood from the context, we sometimes use ρ to refer to the pre-snapshot σ. In this case, if A is an attribute of R, then $\rho.A$ is used to refer to the value of $\sigma(A)$. If attribute A has type $\mathbf{ID}_{R'}$ for some artifact type R' with attribute B, then $\rho.A.B$ refers to the B-value of the (pre-)instance identified by $\rho.A$. More generally, *path expressions* can be constructed that correspond to arbitrarily long chains of this kind of reference.

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 σ be an instance pre-snapshot of artifact type R with ID ρ. The GSM Invariants are specified as follows.

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

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

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

The third invariant stems from the intuition that milestones serve as alternative ways that a stage may be closed. This invariant is typically enforced in practice by syntactic properties of the milestone achieving sentries. The first two are enforced as part of the operational semantics below.

An artifact instance *snapshot* of type R is an instance pre-snapshot σ of type R that satisfies the three GSM Invariants.

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

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

- **Distinct ID's:** If σ and σ' are distinct artifact pre-instance snapshots occurring in the image of Σ, then $\sigma(\mathit{ID}) \neq \sigma'(\mathit{ID})$.
- **No dangling references:** If an ID ρ of type $\mathbf{ID}_R$ occurs in the value of a non-ID attribute of some pre-snapshot in $\Sigma(R')$ for some R' in Γ, then there is a pre-snapshot σ in $\Sigma(R)$ such that $\sigma(\mathit{ID}) = \rho$.

Finally, a *snapshot* of Γ is a pre-snapshot Σ of Γ such that each artifact instance pre-snapshot in the image of Σ is an instance snapshot.

Let Γ be a GSM model and Σ a pre-snapshot of Γ. We now extend the function Σ to ID's and path expressions in the natural manner (e.g., $\Sigma(x.<\mathit{path}>.A[x/\rho])$ will evaluate to the value of attribute A for the artifact instance identified by the path $\rho.<\mathit{path}>$ when followed in (pre-)snapshot Σ).

In application, an *Artifact Service Center* (*ASC*) is used as a container for the set of artifact instances of the artifact types in Γ. 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.

3.4 Events and Tasks

As mentioned informally in Subsection 2.3, there are three kinds of *incoming* events (1-way message, 2-way service call return, and artifact instance creation request), and three kinds of *generated* events (1-way message, 2-way service call, and service call return for artifact instance creation request). These events are represented in the formalism as *messages* that have types, payload signatures, and a unique artifact type as target. For a *ground* event, the main part of the payload is a sequence $(A_1{:}c_1, \ldots, A_n{:}c_n)$ where, for each $i \in [1..n]$, A_i is a data attribute of the target type and c_i is a value of appropriate type. Depending on the message type there may also be payload attributes to help with correlation of service call returns. In practice, the message payloads may also include fault information.

Speaking informally, incoming events are received by the sentries associated with guards and milestones. Generated events are created by *tasks* contained in atomic stages. In particular, there are task types for generating 1-way messages (when invoked they wait for a "handshake" from the ASC indicating success or failure), generating 2-way service calls (when invoked they wait for the ASC to provide the service call return from the called service or a time-out

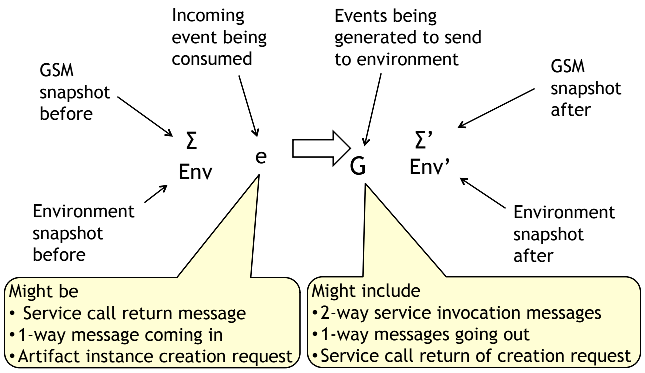

Incoming event being consumed
Events being generated to send to environment
GSM snapshot before
GSM snapshot after
Σ
Env
e
G
Σ'
Env'
Environment snapshot before
Environment snapshot after
Might be
• Service call return message
• 1-way message coming in
• Artifact instance creation request
Might include
• 2-way service invocation messages
• 1-way messages going out
• Service call return of creation request

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

message), and requesting the creations of a new artifact instance (which waits for a service call return with the ID of the newly created artifact instance).

3.5 The Immediate Effect of an Incoming Event

Let Σ be a snapshot, e a ground incoming event, and t a logical timestamp greater than all logical timestamps occurring in Σ. The ASC determines the set of artifact instances in Σ that are *directly affected* by e, using an event-specific query against the family of active artifact instances. For 2-way service call returns the correlation information in the message payload is used to identify the relevant artifact instance. 1-way messages might impact multiple artifact instances.

The *immediate effect* of e on Σ at time t, which is denoted as $\mathit{ImmEffect}(\Sigma, e, t)$, is the pre-snapshot that results from incorporating e into Σ, 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 cause any firing of guard or milestone sentries; this is addressed by the notion of B-step, presented next.

3.6 GSM Business Steps and Logical Timestamps

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 Σ of a GSM model Γ. This is illustrated in Figure 4. In this figure and the next, the "environment" is assumed to include data resulting from artifact-to-artifact messages that is held by the ASC. The semantics characterizes 5-tuples of the form $(\Sigma, e, t, \Sigma', \mathit{Gen})$, where the following hold.

1. Σ is the *previous* snapshot.
2. e is a ground occurrence of an incoming event type associated with Γ.
3. t is a logical timestamp which is greater than all logical timestamps occurring in Σ.
4. Σ' is the *next* snapshot.

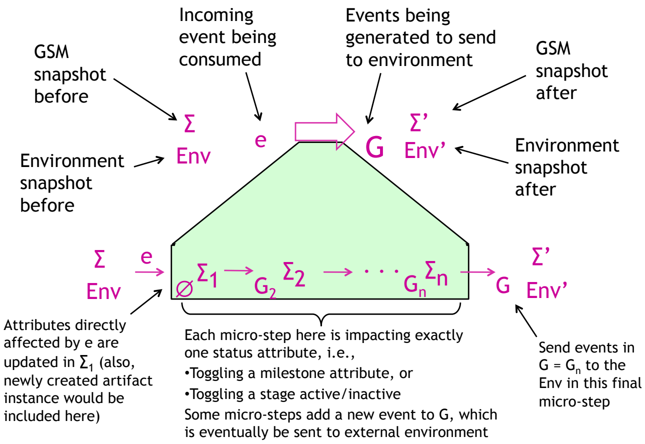

GSM snapshot before
Incoming event being consumed
Events being generated to send to environment
GSM snapshot after
Σ
Env
e
G
Σ'
Env'
Environment snapshot before
Environment snapshot after
Σ
Env
e
∅
Σ1
G2
Σ2
. . .
Gn
Σn
G
Σ'
Env'
Attributes directly affected by e are updated in Σ1 (also, newly created artifact instance would be included here)
Each micro-step here is impacting exactly one status attribute, i.e.,
•Toggling a milestone attribute, or
•Toggling a stage active/inactive
Some micro-steps add a new event to G, which is eventually be sent to external environment
Send events in G = Gn to the Env in this final micro-step

Figure 5: Incremental formulation of GSM semantics

5. *Gen* is the set of ground *generated event occurrences*, all of whose types are outgoing event types associated with Γ.

To illustrate the notion of B-step, we describe key aspects of the incremental formulation of the operational semantics. In this case, Σ' is constructed in two phases (see Figure 5). The first is to incorporate e into Σ, by computing $\mathit{ImmEffect}(\Sigma, e, t)$. (If $\mathit{ImmEffect}(\Sigma, e, t) = \Sigma$ 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 Γ (Subsection 4.2). The second phase builds a sequence

$$\Sigma = \Sigma_0, \Sigma_1 = \mathit{ImmEffect}(\Sigma, e, t), \Sigma_2, \ldots, \Sigma_n = \Sigma'$$

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 Σ_n. (There are restrictions on the ordering of rule application, as detailed in Subsection 5.1.) Here Σ' 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 Σ' and *Gen* from Σ 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.

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 artifact instance, which remain fixed once $\mathit{ImmEffect}(\Sigma, e, t)$ 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: 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 expressions of the form $x.M$ (for 1-way message type M), $x.F^{\mathit{return}}$ (for service call return from F), and $x.create_R^{call}$ (which is a call to create an artifact instance of type R).

- **Internal event expression** (also known as **status change event expression):** This includes
  1. $+\tau.m$ and $-\tau.m$, where τ is a well-formed path expression of form $x.<path>$ with type $\mathbf{ID}_{R'}$ for some artifact type R' in Γ, and where m is a milestone of type R'. Intuitively, an event occurrence of type $+\tau.m$ $[-\tau.m]$ arises whenever the milestone m of the instance identified by $x.<path>$ changes value from false to true [true to false, respectively].
  2. $+\tau.active_S$ and $-\tau.active_S$, where τ is a well-formed path expression of form $x.<path>$ with type $\mathbf{ID}_{R'}$ for some artifact type R' in Γ, and where S is a stage name of type R'. Intuitively, an event occurrence of type $+\tau.active_S$ $[-\tau.active_S]$ arises whenever the stage S of the instance identified by $x.<path>$ changes value from closed to open [open to closed, respectively].

Definition: A *sentry* for artifact type R is an expression $\chi(x)$ having one of the following forms, where x is the context variable of R: "**on** $\xi(x)$ **if** $\varphi(x)$", "**on** $\xi(x)$", or "**if** $\varphi(x)$", and where the following hold.

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

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

Expression $\xi(x)$, if it occurs, is called the (*triggering*) *event*. Expression $\varphi(x)$, if it occurs, is called the *condition*.

We now consider what it means for a pre-snapshot Σ to *satisfy* a sentry χ at time t, denoted $(t, \Sigma) \models \chi$. Satisfaction of a condition by Σ is straightforward, and not considered further. If χ involves an incoming event type E, the expression $\rho.E$ for artifact instance ρ is true if $\rho.currEventType = E$ and $\rho.currEventTime = t$. Similarly, if χ involves a status change event of form $\odot\rho.\tau.s$ (for polarity $\odot$, path expression τ, and status attribute s), the event expression is considered true if the value of $\rho.\tau.s$ matches the polarity $\odot$ and $\rho.\tau.s^{mostRecentUpdate} = t$.

4. PAC RULES AND POLARIZED DEPENDENCY GRAPHS

This section introduces two pillars of GSM. First is a family of ECA-like rules, called "Prerequisite-Antecedant-Consequent (PAC)" rules (Subsection 4.2). Second is the notion of "Polarized Dependency Graph (PDG)" (Subsection 4.3), which is used to define the well-formedness condition on GSM models, and to provide a form of stratification for the application of PAC rules.

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 (Σ, e, t, Σ'), if Σ' is constructed from (Σ, e, t) through the incremental application of PAC rules, then each status value attribute can change at most once during that construction.

Change Dominates Principle: In a B-step (Σ, e, t, Σ'), 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 Σ', 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 Σ'), rather than being hidden in the internal processing that computes Σ'. 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 $\Sigma = \Sigma_0, \mathit{ImmEffect}((, \Sigma, ,)e, t) = \Sigma_1, \ldots, \Sigma_i$. The other context is that of the fixpoint formulation, which focuses on the completed B-step (Σ, e, t, Σ'). 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 Σ in both contexts. It may be thought of as a prerequisite for determining whether the rule is relevant to (Σ, e, t).

Antecedent: This part of the rule is considered relative to Σ_i in the incremental formulation, and relative to Σ' 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 antecedant 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 Σ_i, then the rule is considered to be *eligible*, and it may be *fired* to create Σ_{i+1} according to the consequent. In the fixpoint formulation, if the rule is relevant and Σ' satisfies the antecedent, then Σ' 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 Σ' rather than Σ. Intuitively, the focus is on creating Σ' to be the fixpoint, in the spirit of logic programming, of applying the PAC rules to $\mathit{ImmEffect}(\Sigma, e, t)$. In logic programming, the fixpoint itself satisfies all of the if-then rules, considered as first-order logic formulas. Similarly, in GSM the fixpoint Σ' satisfies the AC part of each PAC rule.

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

First, consider 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

	Basis	Prerequisite	Antecedent	Consequent
Explicit rules				
PAC-1	Guard: if **on** $E(x)$ **if** $\varphi(x)$ is a guard of S. (Include term $x.active_{S'}$ if S' is parent of S.)	$\neg x.active_S$	**on** $E(x)$ **if** $\varphi(x) \wedge x.active_{S'}$	$+x.active_S$
PAC-2	Milestone achiever: If S has milestone m and **on** $E(x)$ **if** $\varphi(x)$ is an achieving sentry for m.	$x.active_S$	**on** $E(x)$ **if** $\varphi(x)$	$+x.m$
PAC-3	Milestone invalidator: If S has milestone m and **on** $E(x)$ **if** $\varphi(x)$ is an invalidating sentry for m.	$x.m$	**on** $E(x)$ **if** $\varphi(x)$	$-x.m$
Invariant preserving rules				
PAC-4^{simp}	Opening stage invalidating milestone: If S has milestone m.	$x.m$	**on** $+\, x.active_S$	$-x.m$
PAC-5	If S has milestone m.	$x.active_S$	**on** $+\, x.m$	$-x.active_S$
PAC-6	If S is child stage of S'.	$x.active_S$	**on** $-\, x.active_{S'}$	$-x.active_S$

(a) PAC rules for Γ, "simplified" version

	Basis	Prerequisite	Antecedent	Consequent
PAC-4	Guard invalidating milestone: If S has milestone m and has guard **on** $E(x)$ **if** $\varphi(x)$ of S, where $E(x)$ is not $-x.m$, and where $\neg x.m$ does not occur as a top-level conjunct in $\varphi(x)$. (Include term $x.active_{S'}$ if S' is parent of S.)	$x.m$	**on** $E(x)$ **if** $\varphi(x) \wedge x.active_{S'}$	$-x.m$

(b) The "enhanced" rule template for PAC-4, which helps to maintain Invariant GSM-1.

Figure 6: Prerequisite-Antecedent-Consequent (PAC) rule templates associated with a GSM model Γ

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 $x.active_{S'}$ 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 Σ' from (Σ, e, t), it is possible that both S' and S are closed in Σ, that some incremental step opens S', and that a subsequent incremental step opens S. In the final result Σ', 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^{simp} and PAC-4, which are focused on Invariant GSM-1. PAC-4^{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^{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 Γ 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 [1, 10]. 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.

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

- 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, node $+R.g$

The set E_Γ of edges for *PDG*(Γ) is based largely on the rules in Γ_{PAC}. In the following, R, R' range over not necessarily distinct artifact types in Γ; s, s' range over not necessarily distinct status attributes of those types; and "$\odot, \odot'$" correspond to polarities, that is, they range over $\{+, -\}$. Let x be the context variable for R. Also, in expression $\odot' \tau(x).s'$, $\tau(x)$ evaluates to ID's of type R'.

- Suppose that (π, α, γ) is a PAC rule in Γ_{PAC} having the form of PAC-2, PAC-3, PAC-5, or PAC-6.
  – If α includes as a triggering event the expression $\odot' \tau(x).s'$ and γ is $\odot x.s$, then include edge $(\odot' R'.s', \odot R.s)$.
  – If α includes in its condition an expression $\tau(x).s'$ and γ is $\odot x.s$, then include edges $(+R'.s', \odot R.s)$ and $(-R'.s', \odot R.s)$.
- Suppose that (π, α, γ) is a PAC rule in Γ_{PAC} having the form of PAC-1, that is created because of guard g for stage S in type R.
  – If α includes as a triggering event the expression $\odot' \tau(x).s'$, then include edge $(\odot' R'.s', +R.g)$.
  – If α includes in its condition an expression $\tau(x).s'$, then include edges $(+R'.s', +R.g)$ and $(-R'.s', +R.g)$.
- Suppose that (π, α, γ) is a PAC rule in Γ_{PAC} having the form of PAC-4, that is created because of guard g and milestone m for stage S in type R.

– If α includes as a triggering event the expression $\odot'\tau(x).s'$, then include edge $(\odot' R'.s', -R.m)$.

– If α includes in its condition an expression $\tau(x).s'$, 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 $(+R.g, +R.\mathit{active}_S)$.

Definition: A GSM model Γ is *well-formed* if $\mathit{PDG}(\Gamma)$ 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 ρ of type R) should be considered before any rule that might use $\rho'.g$ to open its stage (for any ρ' of type R').

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 Γ and E is constructed in the same manner as $\mathit{PDG}(\Gamma)$, except that a rule (π, α, γ) is not considered if π 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 presents the equivalence theorem. The section also considers B-steps in series.

5.1 The Incremental Formulation

Assume that GSM model Γ is given, and let us focus on incorporating event e into snapshot Σ at time t. Recall from Subsection 3.6 and Figure 5 that the incremental formulation is based on the construction of a sequence

$$\Sigma = \Sigma_0, \Sigma_1 = \mathit{ImmEffect}(\Sigma, e, t), \Sigma_2, \ldots, \Sigma_n = \Sigma'$$

(where $\Sigma_1 \neq \Sigma$).

Given Σ_j, $j \geq 1$, a ground PAC rule (π, α, γ) is *applicable* to (or *eligible* to fire with) Σ_j if $\Sigma \models \pi$ and $\Sigma_j \models \alpha$. *Applying* (or *firing*) such a rule would yield a new pre-snapshot Σ_{j+1}, that is constructed from Σ_j by "applying" the effect called for by γ (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 $\mathit{PDG}(\Gamma)$, i.e., for each pair r, r' of ground rules with abstract actions $\odot R.s$ and $\odot' R'.s'$, respectively, if $\odot R.s < \odot' R'.s'$ then the rule r must be considered for firing before the rule r' is considered for firing.

LEMMA 5.1: *Suppose that (Σ, e, t) is a snapshot, ground event, and time greater than all times in Σ. Suppose further that $\Sigma_1 = \mathit{ImmEffect}(\Sigma, e, t)$. Then there is at least one snapshot Σ' obtained by firing the rules of Γ_{PAC} in an ordering that complies with $\mathit{PDG}(\Gamma)$, and where no more rules can be fired. Furthermore, if Σ' and Σ'' are constructed in this manner, then $\Sigma' = \Sigma''$.*

Proof (sketch): The existence of at least one Σ 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 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 Σ' and Σ'' are different end results, and let $\odot\rho.s$ be a least ground status attribute that only one of Σ' or Σ'' changes, where ρ is of type R. Suppose without loss of generality that Σ' is the one where $\rho.s$ changes. Since Σ', Σ'' agree on all of the ground nodes that correspond to the abstract nodes preceding $\odot R.s$ in $\mathit{PDG}(\Gamma)$, the rule that triggered the change to $\odot\rho.s$ in Σ' is also applicable in Σ'', and could thus be fired there, yielding a contradiction. □

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

5.2 The Fixpoint Formulation

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

Intuitively, the first property states that Σ' must comply with all of the demands of the PAC rules.

Definition: Given Γ and (Σ, e, t) as above, with non-trivial immediate effect, then snapshot Σ' is *compliant* with respect to Γ and (Σ, e, t) if

- Σ' and $\mathit{ImmEffect}(\Sigma, e, t)$ agree on all data attributes, and
- for each ground PAC rule (π, α, γ) of Γ_{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 Σ and Σ', then that toggling must be "justified" by some ground PAC rule.

Definition: Given Γ and (Σ, e, t) as above, with non-trivial immediate effect, then snapshot Σ' is *inertial* with respect to Γ and (Σ, e, t) if the following holds for each artifact instance ID ρ in $\Sigma_1 = \mathit{ImmEffect}(\Sigma, e, t)$ 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 (π, α, γ) of Γ_{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 γ.

Definition: A tuple (Σ, e, t, Σ') *satisfies* the fixpoint formulation if Σ' is compliant and inertial with respect to Γ and (Σ, 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" [14].

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

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

$$((\neg R.s \ \wedge \ \bigvee_{(\pi,\alpha,+R.s)\in \mathit{Cnsq}(+R.s)}(\pi \ \wedge \ \alpha')) \rightarrow R.s') \wedge$$
$$((\neg R.s \ \wedge \ \bigwedge_{(\pi,\alpha,+R.s)\in \mathit{Cnsq}(+R.s)} \neg(\pi \ \wedge \ \alpha')) \rightarrow \neg R.s')$$

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

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

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

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

5.4 The Equivalence Theorem

The equivalence of the three formulations of the GSM semantics holds for all GSM models Γ such that *PDG*(Γ) is acyclic.

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

- (Σ, e, t, Σ') *satisfies the incremental formulation.*
- (Σ, e, t, Σ') *satisfies the fixpoint formulation.*
- (Σ, e, t, Σ') *satisfies the closed-form formulation.*

There is exactly one Σ' that satisfies these properties.

Proof (sketch): The second two formulations are equivalent because Ψ_Γ captures in extended FOL precisely the conditions of compliant and inertial. Let Σ' 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. □

5.5 B-steps in Series

This subsection briefly considers situations where 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 b_1 and is closed in some subsequent B-step b_2. Because the assignment is purely computational, it makes sense to have b_2 happen immediately after b_1. The same is true if B-step b_1 generates a message to the ASC intended for another artifact instance, or that calls for creation of an artifact instance, and b_2 processes that message. In practice, we define a *macro-B-step* to be a family of B-steps that starts with incorporation of an incoming event from the environment, and includes any subsequent B-steps stemming from automated actions within the BSC. Macro-B-steps are not guaranteed to terminate, nor to be unique. (We also note that in some corner cases, a change made by one B-step may be "undone" by another B-step in the same macro-B-step.)

6. RELATED WORK

The GSM approach draws on previous work on ECA systems (e.g., [16]), but develops a variant useful for data-centric management of business operations and processes.

There is a strong relationship between the business artifact paradigm and Case Management [21, 8, 22]. Both are data-centric, and support *ad hoc*, constrained styles for managing what activities should be performed and when. In both [21] and GSM, models are defined by adorning activities with a form of pre- and/or post-conditions, and the operational semantics is based on ECA rules derived from them. GSM appears to be more general than [21], and incorporates an explicit milestone construct. The approaches of [21] and GSM may help to provide formal foundations for Case Management systems.

The AXML Artifact model [15] supports a declarative form of artifacts based on Active XML. 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 [20] is a fully declarative business process language, in which the possible sequencings of activities are governed entirely by constraints expressed in a temporal logic. Condition-Response-Graph structures [11] support a family of intuitive constructs that are somewhat related to the GSM constructs, but with formal semantics defined using DecSerFlow. Neither system incorporates data with the prominence that GSM does. GSM does not attempt to support the level of declarativeness found in DecSerFlow, but instead relies on ECA-like rules and a fixpoint characterization.

There is a loose correspondence between the artifacts approach and proclets [19]. 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. GSM places places more emphasis on data, and permits conditions against multiple artifact instances as a declarative form of communication between them.

7. CONCLUSIONS

This paper extends on-going work in the general area of event-driven, declarative, data-centric business process management. Building on two previous papers that introduce the Guard-Stage-Milestone (GSM) approach for specifying business artifact lifecycles, this paper describes how GSM supports the interaction between artifact instances in a declarative manner. The overall behavior of a GSM system is specified in terms of ECA-like rules. These rules are derived from the guards and milestones of stages, providing a natural modular structure for the rules. A precise operational semantics is given, and three formulations for that semantics are shown to be equivalent.

Research currently being pursued includes transactional properties, enabling stages to operate on members of a collection attribute, enabling a stage to operate on multiple artifact instances, and incorporating people and roles.

Acknowledgements

The authors thank the people in the extended Project ArtiFact team, the ACSI team, and several others. Most notable among these are: Diego Calvanese, David Cohn, Giuseppe De Giacomo, Nirmit Desai, Alin Deutsch, Amit Fisher, Nanjangud C. Narendra, Jianwen Su, John Vergo, and Victor Vianu.

8. REFERENCES

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

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

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

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

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

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

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

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

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

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

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

[12] R. Hull et al. Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *Proc. of 7th Intl. Workshop on Web Services and Formal Methods (WS-FM 2010), Revised Selected Papers*, Lecture Notes in Computer Science. Springer, 2010.

[13] R. Hull et al. A formal introduction to business artifacts with guard-stage-milestone lifecycles, Version 0.8, May, 2011. Draft IBM Research internal report, available at `http://researcher.watson.ibm.com/researcher/view_page.php?id=1710`.

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

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

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

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

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

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

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

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

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