The Guard-Stage-Milestone (GSM) Framework for Modeling Artifact-centric Workflows

Riccardo De Masellis

SAPIENZA Università di Roma
Dipartimento di Ingegneria Informatica, Automatica e Gestionale

Seminars in Software and Services
2012-2103
What is an Artifact?

Definition (Artifact)

“It is a data record, evolving over time, representing a business relevant entity”

Introduced by IBM as the core building block for workflows.

**Activity-Centric** approach

Focuses on processes:

**Artifact-Centric** approach

Focuses on data as well as processes:

**Shopping Cart Artifact:**

<table>
<thead>
<tr>
<th>Data Attributes</th>
<th>Status Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomerID</td>
<td>CartID</td>
</tr>
<tr>
<td>@c1</td>
<td>@u3</td>
</tr>
</tbody>
</table>

---

R. De Masellis

The Guard-Stage-Milestone (GSM) Framework for Modeling Artifact-centric Workflows
Consider the process where a *customer* purchases products (called *line items*) from an e-commerce website.

Assuming that the customer is already logged in, the process is composed by the following steps:

- the customer adds one or more *line items* to the *shopping cart*;
- the customer sends the order;
- the company processes the payment and the shipping of the order.
Example (E-commerce process modeled with UML activity diagram)

Question:
Where are data?
Example (E-commerce process modeled with UML activity diagram)

Question:
Where are data?

Answer:
Outside the model (not explicitly modeled)!
Definition (Artifact type)

An artifact type is a tuple $A = \langle D, L \rangle$ where:

- $D$ is called data schema and it captures the artifact's relevant information;
- $L$ is the lifecycle schema which specifies the evolution of the artifact.
Definition (Information Model)

A data schema for artifact type $A$ is a set of attributes.

An artifact instance of type $A$ has a value for each attribute.

Example: Information model of the shopping cart artifact

$$\{ \langle \text{CartId}, \odot c1 \rangle, \langle \text{CustomerID}, \odot u3 \rangle, \langle \text{LineItems}, \{\odot li1, \odot li2, \odot li3, \odot li5\} \rangle, \langle \text{TotalCost}, 350 \rangle, \ldots \}$$
The lifecycle of a process describes the way it evolves over time.
The lifecycle of a process describes the way it *evolves over time*.

It can be specified:

- in a **imperative** way: in each state of the system, the possible ways it can evolve are *explicitly* asserted;
  - **pro**: easy to understand;
  - **con**: hard to specify and modify.
The lifecycle of a process describes the way it *evolves over time*.

It can be specified:

- **in a imperative way**: in each state of the system, the possible ways it can evolve are *explicitly* asserted;
  
  *pro*: easy to understand;
  
  *con*: hard to specify and modify.

- **in a declarative way**: hi-level constraints *restrict* its possible evolutions.
  
  *pro*: hard to understand;
  
  *con*: easy to specify and modify.
Definition (Imperative lifecycle schema)

A imperative lifecycle schema for an artifact type $A$ is a transition system $L = \langle S, s_0, R, T \rangle$ where:

- $S$ is a set of states;
- $s_0 \in S$ is the initial state;
- $R$ is a set of event-condition-action (ECA) rules;
- $T \subseteq S \times R \times S$ is the transition relation.

Example: imperative lifecycle of the shopping cart artifact

For the sake of readability, in the figure we use names for ECA rules.
- **States** represent a “stable” situation in which some business goals has been achieved (possibly after that an action has been performed);
- the **initial state** is the state in which the artifact is when it is created;
- **ECA rules** determine whether the transition has to be performed or not;
- the **transition relation** expresses the relation between states and ECA rules.
States represent a “stable” situation in which some business goals has been achieved (possibly after that an action has been performed);

- the initial state is the state in which the artifact is when it is created;
- ECA rules determine whether the transition has to be performed or not;
- the transition relation expresses the relation between states and ECA rules.

Remark

The transition relation is intended to represent the control flow of the whole process.
Definition (Event-Condition-Action rules)

An event-condition-action rule $\rho$ for an artifact type $A$ is an expression of the form

$$\text{on event if condition then action}$$

where:

- $\text{event}$ is an event type;
- $\text{condition}$ is a boolean formula over data schema of $A$;
- $\text{action}$ is a (possibly complex) activity called a flow.
An event-condition-action rule $\rho$ for an artifact type $A$ is an expression of the form

$$\text{on event if condition then action}$$

where:

- $\text{event}$ is an event type;
- $\text{condition}$ is a boolean formula over data schema of $A$;
- $\text{action}$ is a (possibly complex) activity called a flow.

A flow $f$ for an artifact type $A$ is a (possibly complex) activity whose elements are: atomic tasks, such as assignment or invocation of external services; conditions; cycles; forks and joins.
Remark

We cannot express activities with *pre* and *postconditions*!

Remark

It is always possible to translate an arbitrary complex lifecycle in a *trivial* one, i.e., with a single state and loops where the *process control flow* (i.e., the transition relation) is hidden in the flows.
Declarative Lifecycle (GSM)

**Definition (Declarative lifecycle)**

A declarative lifecycle schema for an artifact type $A$ is a set of triples $\langle G, S, M \rangle$ where:

- $G$ is a boolean formula called **guard**;
- $S$ is called **stage** and it may be atomic or composite. If atomic, it contains a *task*, if composite it is, in turn, a set of $\langle G, S, M \rangle$;
- $M$ is a set of boolean formulas called **milestones**.

**Example: declarative lifecycle of the shopping cart artifact**

[Diagram showing the lifecycle of a shopping cart artifact, with stages such as AddLineItem, createLI, RequestPayment, Payment, and Shipping, and milestones such as LIAdded, Order Sent, Payment Done, ShippedTo, and ShippedTo Customer.]
Declarative Lifecycle

- **Stages** represent either atomic task (or set of tasks) organized in an implicit control flow;
- Guards control the *activation* of tasks: when the guard $g$ of a stage $s$ becomes true, then an activation of task of $s$ starts (and we say that $s$ is “open”);
- Milestones represents goals to achieve. When a milestone $m$ of a stage $s$ is achieved, $s$ “closes”, i.e., the task(s) inside it cannot be performed anymore.
■ **Stages** represent either atomic task (or set of tasks) organized in an implicit control flow;

■ **Guards** control the *activation* of tasks: when the guard $g$ of a stage $s$ becomes true, then an activation of task of $s$ starts (and we say that $s$ is “open”);

■ **Milestones** represents goals to achieve. When a milestone $m$ of a stage $s$ is achieved, $s$ “closes”, i.e., the task(s) inside it cannot be performed anymore.
- **Stages** represent either atomic task (or set of tasks) organized in an implicit control flow;

- **Guards** control the *activation* of tasks: when the guard $g$ of a stage $s$ becomes true, then an activation of task of $s$ starts (and we say that $s$ is “open”);

- **Milestones** represents goals to achieve. When a milestone $m$ of a stage $s$ is achieved, $s$ “closes”, i.e., the task(s) inside it cannot be performed anymore.
Definition (Tasks)

A task is an invocation to an atomic external service specified in terms of preconditions and postconditions.
Definition (Tasks)

A task is an invocation to an atomic external service specified in terms of preconditions and postconditions.

Example (Specification of CreateLI task)

\textbf{CreateLI}() : void  
\texttt{pre} : \top  
\texttt{post} : a new instance \textit{li} of line item artifact type is created and it is added to \texttt{this.LineItems}.  

Guards and milestones are boolean formulas called sentries.
Guards and milestones are boolean formulas called sentries.

**Definition (Sentry)**

A *sentry* is a boolean formula of the form:

\[ \text{on event if condition} \]

where:

- *event* is an event type;
- *condition* is a boolean formula over the data schema of $A$.

Either *event* or *condition* may be omitted.
Guards and milestones are boolean formulas called sentries.

Definition (Sentry)

A sentry is a boolean formula of the form:

\[
\text{on event if condition}
\]

where:

- event is an event type;
- condition is a boolean formula over the data schema of $A$.

Either event or condition may be omitted.

Example: a milestone sentry

\[
\text{OrderSent} : \text{on checkoutEvent if LineItems} \neq \emptyset
\]
Events

We have two types of events:

- **Externally generated events**: events coming from the *environment*;
- **Internally generated events**:
  - events towards the *environment*;
  - milestone achieved, stage opens/closes (in GSM) or flow completed (in Siena).

Definition (Event Type)

An event type $E$ is a set of attributes. An event instance of type $E$ will have a value for each attribute.

Example: an instance of event type `checkoutEvent`:

```
{ ⟨EventID, @e1⟩, ⟨CustomerID, @u2⟩, ⟨CartID, @c3⟩, ... }
```

Remark

Events, differently from artifacts, are immutable.
We have two types of events:

- **Externally generated events**: events coming from the *environment*;
- **Internally generated events**:
  - events towards the *environment*;
  - milestone achieved, stage opens/closes (in GSM) or flow completed (in Siena).

**Definition (Event Type)**

An event type $E$ is a set of attributes.

An event instance of type $E$ will have a value for each attribute.

**Example: an instance of event type checkoutEvent:**

\[
\{ \langle EventID, @e1 \rangle, \langle CustomerID, @u2 \rangle, \langle CartID, @c3 \rangle, \ldots \} 
\]
Events

We have two types of events:

- **Externally generated events**: events coming from the environment;
- **Internally generated events**:
  - events towards the environment;
  - milestone achieved, stage opens/closes (in GSM) or flow completed (in Siena).

Definition (Event Type)

An event type $E$ is a set of attributes.

An **event instance** of type $E$ will have a value for each attribute.

Example: an instance of event type `checkoutEvent`:

\[
\{ \langle EventID, @e1 \rangle, \langle CustomerID, @u2 \rangle, \langle CartID, @c3 \rangle, \ldots \}\]

Remark

Events, differently from artifacts, are immutable.
The environment represents the external world, namely everything that somehow interacts (through external events) with artifacts, like human users and external services (e.g., google maps).

Artifacts communicate with the environment through environment gateways and they receive externally generated event from them.
Environment

The environment represents the external world, namely everything that somehow interacts (through external events) with artifacts, like human users and external services (e.g., google maps).

Artifacts communicate with the environment through environment gateways and they receive externally generated event from them.

**Definition (Environment Gateway Type)**

An environment gateway type \( V \) is a set of attributes.

An environment gateway instance of type \( E \) will have a value for each attribute.

**Remark**

Data of externally generated event instances are taken from environment gateway instances.
Environment

Example (Example)

In our e-commerce example, we have two environment gateway types: the customer and the company.
Example (Example)

In our e-commerce example, we have two environment gateway types: the *customer* and the *company*.

Remark

We cannot neither *directly* modify data of environment gateways nor control *events* generated by them.
Recap

The framework’s building blocks are:

- **artifact** types;
- **environment gateway** types;
- **externally generated event** types;
- **internally generated events**;

---

Environment

Artifacts

Customer \@u1

Customer \@u2

Company \@Amazon

Cart \@c1

Cart \@c3

LineItem \@li1

LineItem \@li4

Internally generated events

Externally generated events
Artifact-based System

Definition (Artifact-based System)

An artifact base system is a tuple $\mathcal{M} = \langle \mathcal{A}, \mathcal{V}, \mathcal{E} \rangle$ where:

- $\mathcal{A}$ is a set of artifact types;
- $\mathcal{V}$ is a set of environment gateway types;
- $\mathcal{E}$ is a set of event types.
Example (Shopping Cart artifact type)

Information Model: \{cartID, CustomerID, LineItems, TotalCost\}

AddLineItemGuard: on chooseItem; LIAdded: on createLICompleted;
OrderSent: on checkoutEvent if LineItems ≠ ∅;
ProcessingGuard: on + OrderSent;
RequestPaymentGuard: on + OrderSent if ¬PaymentDone;
PaymentDone: on PaymentCompleted ShippingGuard: on + PaymentDone;
ShippedToCustomer: on ShippingToCustomerCompleted;
Shipped: on + ShippingToCustomer if PaymentDone.
## Example (Environment gateway and event types)

**Environment gateways**
- **Customer**: \{\text{CustomerID, preferences}\};
- **Company**: \{\text{CompanyID}\}.

**External events types**
- **ChooseItem**: \{\text{CustomerID, cartID, ItemID}\};
- **Checkout**: \{\text{CustomerID, cartID}\}. 

From a declarative lifecycle we extract a set of **PAC rules** (prerequisite - antecedent - consequent rules) which restrict the way the system evolves.

For example, for each stage $s$, and for each guard $g$ of $s$, we have the following rule:

- **Prerequisite**: $s$ is not active;
- **Condition**: $g$ is true;
- **Consequent**: $s$ is active.

In order to understand how such rules are evaluated, we need to introduce the semantics of the system.
The **semantics** of a GSM system (formally) specifies how it evolves.
The semantics of a GSM system (formally) specifies how it evolves.

**Definition (Snapshot)**

A snapshot $s$ of a GSM system $M$ is the state in which $M$ is at a given time, and it is made up by all relevant information for $M$, precisely:

- artifact instances, with data and info about lifecycles (open stages, milestone achieved, etc.);
- the event just consumed;
- the events’ queue (both internally and externally generated).
From the initial snapshot:

1 \textbf{do forever}: 
2 \hspace{1em} \textit{pick} and \textit{consume} an externally generated event $ext_i$ (i.e., (a) copy its payload in the instance and (b) fire the applicable PAC rules); 
3 \hspace{1em} \textbf{while} the \textit{internally generated events’} queue is not empty: 
4 \hspace{2em} \textit{pick} and \textit{consume} an internal event $int_j$ (i.e., (a) change data according to the event and (b) fire the applicable PAC rules);
From the initial snapshot:

1 **do** **forever**:
2    *pick and consume* an externally generated event $ext_i$ (i.e., (a) copy its payload in the instance and (b) fire the applicable PAC rules);
3    *while* the internally generated events’ queue is not empty:
4    *pick and consume* an internal event $int_j$ (i.e., (a) change data according to the event and (b) fire the applicable PAC rules);

$$S_0 \xrightarrow{ext_1} S_1 \xrightarrow{int_1} S_3 \xrightarrow{int_2} \ldots \xrightarrow{int_n} S_n \xrightarrow{ext_2} S_{n+1} \xrightarrow{int_{n+1}} \ldots$$

represent a possible system **run**.
Example (E-commerce possible run)

Suppose we have an artifact instance $c$ of *ShoppingCart* type and one gateway type instance $u$ of type *Customer*.

1. The event $e_1$ of type *ChooseItem* is picked and its consumption results in switching the *AddLineItemGuard* to true, opening stage *addLineItem* and invoking the (external) task *createLI*;

2. an internally generated event $e_2$ is produced as the task *createLI* completes;

3. the event $e_2$ is picked and its consumption results in achieving the *LIAdded* milestone, that in turn, produces an event $e_3$ of *milestone achievement*;

4. the event $e_3$ is picked and its consumption and produces the effect of closing stage *AddLineItem* (a PAC rule).

5. another externally generated event is picked . . .

6. . . .
Every run depends on which events we pick...
Evolution of the System

Every run depends on which events we pick...

Given an initial snapshot $s_0$, we can build a transition system representing all possible evolutions of the system!
Example (E-commerce all runs)

- **s0**: Filling, LIAdded
- **e1**: ChooseItem
- **e2**: CreateLICompleted, createLI in execution
- **s1**: Filling, AddLineItem
- **e2**: CreateLICompleted
- **s2**: Filling, LIAdded
- **e3**: LIAddedAchieved
- **s3**: Filling, LIAdded
- **e4**: ...
- **s4**: Filling, LIAdded
- **e2**: Checkout
- **s2**: OrderSent
- **e5**: OrderSentAchieved
- **s2**: OrderSent, Processing, RequestPayment, Payment in execution
- **e6**: PaymentCompleted
- **s2**: OrderSent, Processing, PaymentDone
Conclusions

We build the transition system in order to \textit{explore} it, for \textit{verification} purposes.
We build the transition system in order to explore it, for verification purposes.

**Bad news**
The transition systems has (in general) infinite states!

We cannot rely on “standard” model checking.
Conclusions

We build the transition system in order to explore it, for verification purposes.

Bad news

The transition systems has (in general) infinite states!

We cannot rely on “standard” model checking.

Good news

There are techniques [7, 3, 5, 1] that deal with this issue achieving interesting verification decidability results.
Foundations of relational artifacts verification.
In *BPM*, 2011.

Verification of gsm-based artifact-centric systems through finite abstraction.

Artifact systems with data dependencies and arithmetic.

On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles.


Facilitating workflow interoperation using artifact-centric hubs.