A Generic Business Artifacts Based Authorization Framework for Cross-Enterprise Collaboration

Lior Limonad, David Boaz
IBM Haifa Research Lab, Carmel Mountain, Haifa 31905, Israel
{liorli, davidbo}@il.ibm.com

Richard Hull, Roman Vaculin, Fenno (Terry) Heath
IBM T.J. Watson Research Lab, Hawthorne, New York, USA
{hull, vaculin, theath}@us.ibm.com

Abstract: Business artifacts provide an approach to Business Process Management that combines data and process in a holistic way. Previous research introduced artifact-centric Interoperation Hubs (I-Hubs) as a data-centric alternative to conventional service orchestration for enabling the cooperative interaction of multiple organizations with shared business objectives. The current paper extends this vision by describing an approach for implementing I-Hubs that supports rich access control mechanisms. This reflects the data-process duality of business artifacts, the approach borrows from access control disciplines for both data and process. The paper describes how the approach is being applied in connection with two models for artifact lifecycles, a procedural one based on finite state machines and a declarative one based on guards, stages, and milestones.

Keywords: Business Artifact (aka Business-Entity with Lifecycle), Business Process Management, Cross Enterprise Collaboration, Service Interoperation.

1. Extending Artifact Systems with Access-Control

Web services and support for web service interoperation has emerged as a foundational component in enabling modern enterprises to interact in support of shared business objectives. Citation [1] introduces Artifact-centric Interoperation Hubs (or I-Hubs for short) as a basis for supporting inter-organization interaction that is much more data-centric than conventional approaches to service orchestration. I-Hubs are based on Business Artifacts (or ‘artifacts’ for short) [2], [3], an approach to Business Process Management that combines data and process in an holistic way. This paper introduces a novel, generic access-control realization framework for I-Hubs that combines aspects of access control for both database systems and process-based systems. The paper illustrates the generality of the framework by describing how it can be used with two distinct approaches to modeling artifact lifecycles, one procedural and the other declarative. As part of the EU-funded Artifact-Centric Service Interoperation (ACSI) project [4], the framework has been implemented for the procedural lifecycles, and the implementation for the declarative lifecycles is currently underway.

1 The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement no. 257593.

Business artifacts are used to model key business-relevant conceptual entities that progress through business operations. Artifacts are modeled using an Information Schema, that includes attributes for storing all business-relevant information about the artifact as it moves through the business, and a Lifecycle Schema, that represents the possible ways that the artifact might progress through the business. Artifacts provide a cross-cutting, end-to-end perspective across multiple inter-related business processes. More broadly, the artifact approach typically yields a high-level factoring of inter-related business processes into a handful of interacting artifact types. Citation [5] provides further motivation for the artifact approach and references to several applications of it.

An artifact system is a BPM system that enables specification of artifact-types and interaction with artifact instances. The system provides the following runtime services: query and retrieval of artifact information, invocation of business events (which typically result in changes in the overall state of one or more artifact instances), and providing notifications about pre-subscribed events.

A variety of models can be used to specify the lifecycle schemas for artifacts. One natural approach is to use Finite State Machines (FSM) for the lifecycle schemas. This is a variation on the lifecycle models used in the original works on artifacts [6], and is embodied by the Siena prototype system [7]. In this approach, each state corresponds loosely to a “milestone”, i.e., a business-relevant operational objective that an artifact instance might achieve. Transitions between states are triggered by explicit requests from the environment formed as business events. A more declarative approach, called Guard-Stage-Milestone (GSM) [8–10], also focuses on milestones, but allows that multiple milestones may be true at the same time. This corresponds to the intuition that the achieving of milestones is often cumulative, e.g., at one point in time, a budget approval might be obtained and at another point in time a legal approval might be obtained; and both approvals may be needed before the start of some manufacturing process. In GSM, “stages” correspond to clusters of activities that are intended to achieve one or more milestones, and “guards” correspond to conditions under which a stage should be opened. GSM uses a variant of Event-Condition-Action (ECA) rules to specify the guards, and also to specify whether milestones should be achieved or invalidated. Stages can be organized into hierarchies, allowing for modularity. The GSM approach is
Access control systems are expected to provide three essential services:

1. **Identification and authentication** - associating users with the software subjects.
2. **Authorization** - determining what a subject can do, e.g., Create-Read-Update-Delete-Append and Execute operation (i.e., CRUDAE).
3. **Accountability** - auditing and logging what a subject did.

A major contribution of the current paper is to develop a rich, robust framework for supporting the access controls needed by I-Hub systems, and showing how this can be implemented as a layer above an underlying artifact engine. Furthermore, the paper illustrates how this framework can be instantiated above finite state machine based artifacts (using Siena) or above GSM-based artifacts (using Barcelona).

Considering the unique inherent dichotomy in artifacts as reflecting both process and data aspects, in this paper we report a novel access-control framework implemented as part of the work in the ACSI project. This effort is aimed to enrich an artifact-system with access-control capabilities combining both the workflow access-control principles and data authorization views.

Organizationally, Section 2 reviews access control techniques for both process-centered and database systems. Section 3 describes the three kinds of access control relevant to artifact-based I-Hub systems. Section 4 introduces the finite state machine and GSM artifact models, and Section 5 describes how an access-control layer can be placed on top of both of these. Section 6 provides some details about the implementation. Section 7 offers brief conclusions.

**II. PROCESS AND DATA ACCESS CONTROL (SOLUTION’S BACKGROUND AND REQUIREMENTS)**

As a first step towards the construction of an artifact-centric access control, in this section we devise high-level requirements for such a mechanism. As explained above, due to the need to ensure such a mechanism governs for both process and data authorization aspects, the requirements have been originated from two sources: workflow access-control and data authorization. Correspondingly, the following subsections detail a set of access-control requirements for these two aspects.

**A. Workflow-access-control**

From a socio-technical perspective, IT in organizations is both affecting and being affected by people and other resources, being assigned the responsibility to safeguard assets and sensitive information, authorize and authenticate identity as part of access control mechanisms, account for information privacy and integrity, and orchestrate activities to improve business operations effectiveness and efficiency. Typically, nowadays IT suites include some form of workflow systems being aimed to facilitate both the design and execution of business processes, either in whole or part [14]. Like any other software deployed within the business environment, workflow systems are expected to manifest typical information security capabilities (e.g., ISO 7498-2). However, some unique aspects have been previously embodied in the Barcelona prototype [11], a descendant of Siena.

An **I-Hub system** is a specialization to an artifact system, aimed to support collaborations between multiple participating organizations. We use the term **participant** to refer to a participating organization in an I-Hub context. The I-Hub holds an artifact-based schema of the business process that is performed jointly by the participants. Different participants are able to access information about the status of relevant artifact instances, invoke changes to the artifact instances, and subscribe to events concerning changes to the artifact instances caused by other participants. Unlike conventional service orchestration systems, which focus primarily on the activity flow governing when services from different participants are invoked, an I-Hub explicitly models the information being maintained and used during the business collaboration, along with the current status of the flow of activity. By maintaining both the data and the process status of a collaborative business activity, the I-Hub holds much more business context than typical orchestration systems; this can be used by the participants to simplify what they need to store internally about the status of their partners. Furthermore, while service orchestration systems are typically pro-active in their control of the participating organizations, an I-Hub system is typically more passive and re-active, permitting organizations more flexibility around when and how they interact with the I-Hub.

The I-Hub approach forms one of key starting points of the EU-funded ACSI research project. A goal of that project is to substantially develop and test the I-Hub framework introduced in [1]. A key component of the project involves extending the Siena and Barcelona prototypes so that they can serve as the artifact-based kernel for I-Hubs.

While some existing artifact-based systems, including Siena, incorporate support for user roles and some forms of access control, these capabilities are blended into the other systems capabilities and are fairly immutable. Furthermore, as originally detailed in [1], the kinds of access control needed to support collaboration between organizations is typically richer than what is needed for roles within a single organization. For example, within a single organization there is typically no concern if a given actor can see the entire process that she is participating in. In contrast, in an I-Hub context, some parts of a business process used by one participant (i.e., participating organization) might need to be hidden from some other participant. Further, I-Hub systems typically follow the fundamental principle of least privilege for access control [12], according to which every participant must be able to access only the objects that are necessary for its legitimate purpose. For example, it is typical that a customer has permission to approve only his own orders.

**Access control** in general is concerned with granting or denying the allowed activities of legitimate users, and mediating every attempt by a user to access a resource in the system [13]. An access control service considers two entity types: the entities that can perform actions in the system (namely, **subjects**), and the entities representing resources to which access may need to be controlled (namely, **objects**).
identified as being particularly essential in such systems [15–
17]. Such requirements have been synthesized in [18], and
have been further adapted for the purpose of this work to
include the following key requirements:

1. **Role-based access control** – in its very core, an access
control system which is aimed to prevent unauthorized
users from either accessing or affecting certain
resources, is typically centered on the concept of a **role**
such that users are clustered by these roles, to which
various **permissions** are being assigned.

2. **Contextual-sensitivity** - workflow systems introduce an
additional degree of complexity to access control, which
is the process’s contextual circumstances in which
resource access attempts are being made (e.g., during
certain tasks or after the granting of other permissions).
Hence, access control enforcement should be also
sensitive to the specification of permissions being
dependent on some ordering criteria associated with the
process execution plan (namely, the process schema).

3. **Separation of duty** – aimed to prevent fraud and error in
organizational settings that involve resources being
shared and accessed by different individuals from
different units, a control system is also expected the
capacity to limit and segregate the extent of
responsibilities each individual is assigned with respect
to a certain resource (and contextual circumstances).
Hence, a control system should also implement some
form of interaction recording to support the capability to
enforce that every individual’s access attempt is in line
with previous interactions (e.g., an attempt to approve a
payment previously submitted by the same person).

### B. Data Authorization

Access control has been an area of interest primarily in the
database community. Traditionally, this has been achieved
using user-specific materialized views that filter out
unnecessary information. However, this basic approach has
been found to mainly suffer scalability issues when there is a
large number of users, and as a result a need to replicate and
maintain a large set of user-specific access policies. As an
alternative, the approach of “authorization views” has been
developed recently [19–22]. An authorization view can be
thought of as a virtual view that specifies what information a
user is allowed to access. However, the user writes the query
in term of the base schema, being oblivious to the
specification of the authorization view, and then access
control determines if the query can be rewritten using the
authorization view. For the purpose of this work we adopt
this approach as an additional requirement.

Hence, we also require:

4. **Authorization-view based access control** – access
control permissions will be specified in a form that enables “on the fly” inference of the validity of each
individual access attempt, rather than pre-computing
participant-specific materialized subsets of accessible
data as in the case of database views.

### III. AN ARTIFACT-CENTRIC ACCESS CONTROL MODEL

Adhering to the requirements of Section 2, the ACSI
implementation for an access-control layer has been
anchored in prior I-Hub literature [1], which illustrates a
unique artifact-centric authorization model that imposes
three complimentary types of access restrictions on the
interactions between participants and an artifact-centric
systems: **views**, **windows**, and ‘**CRUDAE’** runtime access
controls. For each model component listed below, we also
explain its correspondence to the aforementioned
requirements. The first two are coarse-grained and the third
is more fine-grained.

**View** – a component used to restrict what parts of an artifact-
type schema each participant can access. Thus, the
dichotomy in artifacts entails a corresponding view
specification comprising:

1. A specification of a subset of the data attributes which
may be visible. Such restrictions are typically defined
in a form of data attribute **projections**. (Requirement 4)

2. A specification of a portion of the lifecycle behavior,
encapsulating away details in the lifecycle schema that
are not within the execution responsibility of the
participant (Requirements 1 & 2). Such
restrictions are typically attained using a set of
‘abstraction operations’. For example, a common
abstraction operation in the context of finite state
machines is the one of **node condensation**, which
abstracts away lifecycle detail to become invisible in
the participant’s view.

More specific types of restrictions are given below as
concrete examples for the realization of the above
restrictions. With respect to the aforementioned
requirements, this component is well associated with the
contextual-sensitivity requirement, ensuring that each
participant is assigned privileges only within the relevant
contextual circumstances, being determined by the portions
of the lifecycle and data being either exposed or filtered out
in the view’s specification.

**Window** - determines which artifact instances each
participant can access (Requirement 3). For example, in a
sales application the following window filter can be specified
to ensure that participants of type Customer can only see
artifact instances they themselves created: 
**customerID = $[participant].ID** This component supports
the need to ensure segregation of duties, such that specific window
filters may be specified (e.g., ensuring that during runtime,
artifact instances to which access should be prohibited will
not be made visible to the corresponding participants).

‘**CRUDAE’** - determines attribute specific access rights (i.e.,
read, update, append), artifact instance creation/deletion,
and artifact specific service execution eligibility
(Requirement 1). This part of the model maps directly to
the notion of **permissions** to be assigned to different
**roles** as required by the core role-based access
requirement. Complementary to that, our
implementation enables also the specification of participant-filters in order to restrict the exact set of individual users to which any of the authorization model’s components apply and need to be enforced by the access control system.

IV. TWO ARTIFACT-CENTRIC LIFECYCLE MODELS

This section introduces the two artifact-centric models that are being used to implement and validate the artifact-centric access control model introduced above. The procedural Finite State Machine (FSM) model is introduced first, followed by the declarative Guard-Stage-Milestone (GSM) model. The models are illustrated here using a running example that is also used to illustrate the access control constructs. More detailed descriptions of these models can be found in [2], [5], [8], [9].

The running example, called “Custom Manufacture” is based on a manufacturing application, in which customized machines are built on demand, possibly for export. The example involves three (kinds of) participating organizations: Customers that make orders for custom machines, specify high-level requirements, etc.; Manufacturer that builds the machines; and Legal Advisor (who are hired by the Manufacturer) that manage legal requirements concerning the export of the machines. The running example focuses on just one portion of the overall application domain, a portion dealing with establishing the requirements, engineering design, and export considerations for an ordered machine.

A full artifact-based model for Custom Manufacture could involve a handful of artifact types, including Customer Order that manages an overall order, from requirements specification to delivery (and including export controls); Parts Order that manages orders by the Manufacturer; Production Schedule that manages, for each Customer Order, the sequencing of specific manufacturing steps on the factory floor; and possibly some others. In the discussion below the focus is on just part of the Customer Order artifact type, as realized in both the FSM and GSM models.

A. Finite-State-Machine (FSM) Lifecycle Model

An FSM-based artifact type includes both an Information Schema and a Lifecycle Schema. To not restrict the I-Hub framework to any platform-specific data-storage technology, in the current paper, the Information Schema is operationalized simply as a data schema $D_A = a_1 \ldots a_n$ (e.g., a set of attributes), where the values associated to a given attribute may have complex structure. An FSM-based lifecycle schema specifies a finite number of states for a corresponding artifact instance, such that the instance is in only one state at a time. The state the artifact instance is in at any given time is called the current state and can change from one state to another when initiated by a triggering event or condition, this is called a transition. The FSM model is considered procedural because all transitions must be defined explicitly.

More formally, an FSM lifecycle schema is a triple $L = (S, E, A)$, where: $S$ is a finite set of states, which includes the designated states source and sink; $E$ is a set of directed edges (ordered pairs over $S$), designating all possible state transition in a certain artifact type’s lifecycle; and $\lambda$ is a mapping that associates each edge to a non-empty set of external event types that can trigger a transition from the tail of the edge to the head of the edge.

Fig. 1 illustrates part of the FSM-based artifact type for Customer Order. The portion of the artifact lifecycle schema shown focuses on gathering requirements for the manufacture of a new machine, compiling a list of part restrictions (relevant if the machine will be exported), performing engineering design, and finalizing the engineering design. The horizontal line of states corresponds to a “sunny day” execution, and the back arrows correspond to what might happen if the customer requests a change to the requirements. Intuitively, in most cases a transition corresponds to the performance of one or several processing steps. Importantly, all business-relevant information about the progress of an artifact instance is to be stored in the Information Schema.

In terms of the three kinds of participating organizations, the Customer is involved with launching the project and reaching the Requirements Approved state (i.e., transitions 1 and 2). The Customer may also request a change in requirements (i.e., transitions 6, 7, and 8) or to shelve the project (i.e., transitions 10 to 14). The Manufacturer is also involved with reaching the Requirements Approved state (i.e., transition 2), and with the Engineering Design states (i.e., transitions 4, 5, and 9). The Manufacturer can also shelve the project (i.e., transitions 11 to 14). Finally, the Legal Advisor is involved with compiling the Restricted Parts List based on the country the product will be shipped to (i.e., transition 3).

B. Guard-Stage-Milestone (GSM) Lifecycle Model

GSM has been developed in recent years as a declarative approach to specifying the lifecycles of Business Artifacts [8–10]. It is considered declarative because the phase transitions in the lifecycle are not defined explicitly, but rather by Event-Condition-Action (ECA) rules. GSM is suitable for cases where there is a large set of alternate activities for achieving a goal.

As with the FSM case, GSM artifacts have both an Information Schema and a Lifecycle Schema. Fig. 3 shows a specification of the overall GSM model using UML. The main components of a GSM lifecycle schema are milestones and stages. A milestone corresponds to a business-relevant operational objective that may be achieved by an artifact instance. Milestones may also be invalidated, if circumstances arise whereby the operational objective must be re-worked or is otherwise lost (e.g., if the engineering design completed but then the customer changes the underlying requirements). Milestones can also be treated as...
Booleans. A milestone tests to true if it has been achieved and has not since been invalidated; and it tests to false otherwise. A stage represents an atomic or composite activity that is performed to progress an artifact instance through the business. Stages may be nested, permitting hierarchical levels of abstraction. In the original papers on GSM [8–10] each milestone was associated with a stage, and achieving the milestone was a condition for ending the execution of the stage. In the current work this restriction is relaxed, and milestones may also be “freestanding”, i.e., not associated with the termination of any stage.

A key component in GSM-lifecycles are provided by sentries. A sentry is a sensor that is triggered when a predefined circumstance (either an event, or a condition based on the data currently in the information schema, or a combination of an event plus a condition) is detected. The events might be external, or might correspond to changes in the status of a milestone or stage. Each sentry has a possible action associated with it. A guard is a sentry that can open a stage; a terminator is a sentry that can close a stage; an achiever is a sentry that can achieve a milestone; and an invalidator is a sentry that can invalidate a milestone. As noted above, in some case a milestone can serve as the terminator for a stage. Thus, sentries are a specialized form of ECA rules, where the action involves manipulating the status of a stage or a milestone.

Fig. 2 shows part of a GSM-based artifact type for Customer Order. The information model includes status attributes; these include each milestone (considered as a Boolean) and also for each stage S an attribute actS that evaluates to true if stage S is currently active, and evaluates to false otherwise. Turning to the lifecycle diagram, milestones are depicted as circles and stages as rounded-corner rectangles. Several of the milestones have the same names as the corresponding states in the FSM lifecycle of Fig. 1. Importantly in GSM, multiple milestones can be true at the same time. For example, in Fig. 2 the milestones Requirements Approved and Restricted Parts List Compiled may both be true at the same time. The guards are shown using diamonds and the terminators are shown using bowties. (The achievers and invalidators for milestones are typically not shown, but can be, again using diamonds and bowties.) Fig. 2 also shows a transition arrow, from the milestone Restricted Parts List Compiled to a guard of the stage Preparing Import Documents. Such arrows can be viewed as shorthands or macros, that correspond to a guard stating that if the milestone is achieved then the stage should be opened.

During execution of a GSM artifact instance, zero or more stages may be “open” or “active” at a given time. For example, Requirements Gathering and Evaluating Country Restrictions will typically be open in parallel. When a single external event is being processed, a change in status for multiple stages and/or milestones might be triggered. The full set of changes resulting from the processing of an external event is called a Business Step or B-step. References [9], [10] describe the semantics of B-steps in more detail.

We present now the formal description of GSM lifecycle schemas used in this paper; this will be needed when describing the access control framework for GSM. A GSM system is specified as a pair \((M^G, Env)\) where \(M^G\) is an artifact system, i.e., a set of artifact-type specifications (which may refer to each other), and \(Env\) is the environment specified as \((Serv, BizEv)\), where \(Serv\) is a set of callable 1-way and 2-way services; and \(BizEv\) is a set of (external) business event types that can be sent into the artifact system. In the artifact system \(M^G\), each artifact type includes an Information Schema and a Lifecycle Schema. The information schema is analogous to the FSM case. The lifecycle schema is specified by a 12-tuple as follows:

\[
L = \left( S_A, Stages, Milestones, Substages, Sentries, \right.
\]

\[
\left( Achievers, Invalidators, Guards, Terminators, \right)
\]

\[
Tasks, Owns, Transitions \right)
\]

, where:

- \(S_A\) - is a set of Boolean status attributes of this artifact type, that is disjoint from the set of data attributes in \(P_A\).
- There is a status attribute for each milestone, and also a status attribute for each stage (which indicate whether the stage is currently open or closed).
• **Stages** - is a set of stage names, or simply, stages. For each stage \( s \in \text{Stages} \), there is a corresponding stage status value attribute in \( S_s \), whose value corresponds to whether \( s \) is currently active or not.

• **Milestones** - is a set of milestone names, or simply, milestones. For each \( m \in \text{Milestones} \), there is a corresponding milestone status value attribute in \( S_m \).

• **Substages** – is a function from Stages to subsets of Stages, such that the relation \( \{(S,S')|S' \in \text{Substages}(S)\} \) 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.

• **Sentries** – is a set of sentries of this artifact type \( A \). A sentry for artifact type \( A \) is an expression \( \chi \) having one of the following forms:
  - on \( \xi \) if \( \varphi \)
  - on \( \xi \)
  - if \( \varphi \)
where the following holds: If \( \xi \) appears, then it is an event expression for \( A \). If \( \varphi \) appears, then \( \varphi \) is a well-formed formula over the artifact types occurring in \( M^G \). Expression \( \xi \), if it occurs in the sentry, is called the triggering event. Expression \( \varphi \), if it occurs in the sentry, is called the condition.

• **Achievers** - is a function from Milestones to finite, non-empty sets of sentries. For milestone \( m \), each element of \( \text{Achievers}(m) \) is called an achieving sentry of \( m \).

• **Invalidators** - is a function from Milestones to finite sets of sentries. For milestone \( m \), each element of \( \text{Invalidators}(m) \) is called an invalidating sentry of \( m \).

• **Guards** - is a function from Stages to finite, non-empty sets of sentries. For stage \( s \), each element of \( \text{Guards}(s) \) is called a guard of \( s \).

• **Terminators** - is a function from Stages to finite sets of sentries. For stage \( s \), each element of \( \text{Terminators}(s) \) is called a terminator of \( s \).

• **Tasks** - is a function from the atomic stages in Stages to the set of possible task-specifications over \( A \). Such task specifications may include, but are not limited to: value updates to artifact attributes \( D_A \), and 1-way and 2-way service calls (where some 2-way service calls may have the effect of invoking activities to be performed by external human beings).

The above constitute the kernel constructs for specifying GSM lifecycles. The remaining two components are “macros” that are defined in terms of those kernel constructs.

• **Owns** – is a function from Stages to finite subsets of Milestones, such that Owns\((s) \cap \text{Owns}(s') = \phi \) for each \( s \) and \( s' \) a milestone \( m \) if \( m \in \text{Owns}(s) \). Notice that the set Milestones \( \cup_{s \in \text{Stages}} \text{Owns}(s) \) is not necessarily empty, allowing for free-standing milestones. If \( m \in \text{Owns}(s) \) this means that there is a terminator of \( s \) having form “on \( m \).achieved.”

• **Transitions** is a set of edges from milestones to stages. If \( (m,s) \in \text{Transitions} \) this means that there is a guard of \( s \) having form “on \( m \).achieved.”

V. TWO CONCRETE REALIZATIONS OF THE ACCESS MODEL

This section describes concrete realizations of the access model, relative to both the FSM and GSM approaches to specifying artifact lifecycles.

A. Finite-State-Machine Authorization Model

Relating to an FSM-specific lifecycle style, the realizations of the three types of access restriction are as follows.

1) **View\(^{FSM} \):**

An FSM-specific view specifies two types of constraints:

- Restrictions on the subset attributes that can be seen. This restriction is formalized as a set of projection functions \( \pi_j(D_A) \to D'_A \) over the overall set of attributes \( a_1, a_2, \ldots, a_n \) such that a projection mapping over \( D_A \) is an expression of the form \( \pi_j \) where \( j \) is a subset of \( a_1, a_2, \ldots, a_n \) yielding a participant specific data-schema \( D'_A \).
- In FSM, each lifecycle node is considered as a single state in the lifecycle schema. Hence, node condensation specifies the set of lifecycle states that can be seen. This restriction is formalized as a set of condensation functions \( \gamma_f \) over the lifecycle schema \( L \) where \( f:S \to S' \) is a surjective function over \( S \), such that:
  a) \( \text{source, sink} \in S' \),
  b) \( \gamma_f(\text{source}) = \text{source} \) and \( \gamma_f(\text{sink}) = \text{sink} \), and
  c) \( \gamma_f(L) = (S',E') \)
where \( E' = \{(\gamma_f(s_1),\gamma_f(s_2))|s_1,\ldots,s_2 \in E\} \).

Note: multiple states of \( S \) might map into \( \text{source} \) or \( \text{sink} \).

To illustrate this in terms of the Custom Manufacture example introduced in Section 4, the information and processing related to export restrictions might be hidden from Customer participants. In terms of the information schema, this would include hiding attributes that hold information about what parts are prohibited for that Customer. For the FSM lifecycle schema (see Fig. 1), this would include forming a single condensed node from the nodes Requirements Approved and Restricted Parts List Compiled.

2) **Window\(^{FSM} \):**

In order to restrict the set of instances for each participant, an FSM-specific window is formalized as a set of selection functions \( \sigma_p(A) \): \( A^I \to A'^I \) whereas:

- \( A'^I \) is a subset of \( A^I \) of all artifact instances legible to be seen by participant \( p \), and
- A \( \sigma_p(A) \) restriction is defined for all lifecycle states in \( L(A) \).
In terms of the running example, Customer participants are permitted access only to Order instances that they initiated, and a given Legal Services provider would be given access only to Order instances for which they are providing export control services.

3) ‘CRUDA’\textsuperscript{FSM}.

An FSM-specific runtime access control is specified as a set of mapping functions \( \alpha \) between \([ R’, U’, A’ ]\) actions and each attribute in \( D_A \), between \([ C’, D’ ]\) actions and the artifact type , and between the execution action \( E’ \) and state transitions in the lifecycle of the corresponding artifact type, such that:

- \( \alpha( att_i \in D_A ) \rightarrow 2^{[ R’ \cup U’ \cup A’ ]} \)
- \( \alpha( A ) \rightarrow 2^{[ C’ \cup D’ ]} \)
- \( \alpha( E’ ) \subseteq E \) (i.e., the subset of executable transitions in \( L(A) \)).

As one illustration of the ‘E’ restriction in the running example, Legal Services participants are not permitted to invoke transitions that move to the state Project Shelved (i.e., transitions 10 to 14 in Fig. 1). As an illustration of a ‘C’ and ‘D’ restrictions, the Manufacturer is not permitted to create or delete instances of Customer Order.

B. Guards-Stages-Milestones Authorization Model

Relating to a GSM-specific lifecycle style, we can further refine the definitions for all three access restriction types as follows:

1) View\textsuperscript{GSM}.

As in the case of FSM above, GSM-specific views over \( M^G \) are defined in terms of views over the individual artifact types \( A \) in \( M^G \). A GSM-specific view is conceptualized as a set of constraints associated with a relationship between a participant \( p \) and a specific artifact type \( A \) in \( M^G \). Specifically, given an I-Hub system with a corresponding GSM schema \( M^G \), a participant \( p \) is permitted to see a view of \( M^G \) and its snapshots being induced from the specification of the set of view-schemas over all artifact types in \( \mathcal{A} \).

- Let \( A \) be an artifact type of \( M^G \). A view-schema over \( A \), denoted \( V(A) \), is specified as an ordered pair \( (P, C) \), where:
  - \( P \) denotes a restriction over the set of data attributes in \( D_A \) that can be seen, achieved using attribute "projection": Let \( a_1 \ldots a_n = \bigcup_i attr(R_i) \) be the union of all relation attributes in \( D_A \) (i.e., the database schema for artifact type \( A \)); a projection mapping over \( D_A \) is an expression of the form \( \pi_j \) where \( j \) is a subset of \( a_1 \ldots a_n \).
  - \( C \) is a finite set of abstraction operations over \( A \), taken from the following set of operators:
    a) \( condens_{\alpha, \beta}(s) \) – similarly to FSM node-condensation, each lifecycle node in GSM may be considered as a particular stage in the lifecycle schema. Hence, abstracting away particular lifecycle execution detail is equivalent to making the body of a given stage \( s \) including all of its substages and their corresponding owned milestones removed from \( V(A) \).
b) \( \text{hide}_{\text{stage}}(s) \) - makes only the target stage \( s \) removed from \( V^A \) retaining all descendant stages and milestones of \( s \) in \( V(A) \).

c) \( \text{hide}_{\text{milestone}}(m) \) - removes milestone \( m \) from \( V(A) \).

d) \( \text{hide}_{\text{entry}}(s) \) - removes sentry \( s \) from \( V(A) \).

The aforementioned set of operations may be further extended with additional user-specific macro-operations defined as combinations of the above operations. For example, one may define the operation \( \text{eliminate}_{\text{stage}}(s) \) as a combination of: \( \text{condense}_{\text{stage}}(s), \text{hide}_{\text{stage}}(s) \), and an iterative application of \( \text{hide}_{\text{milestone}}(m) \) to all milestones owned by \( s \).

For each participant \( p \), the view induced by view \( V(A) \) is formed from \( A \) as follows:

- The set of data attributes of \( V(A) \) is the projected set of attributes \( P(D_p) \).
- The set of status attributes of \( V(A) \) is the projected set of attributes \( C(S_p) \) implied from the exclusion of all status attributes that correspond to either a stage or a milestone which has been eliminated by any of the operations in \( C \).
- The set of sentries of \( V(A) \) is a subset of \( Sentries(A) \) excluding all sentries included in an operation \( \text{hide}_{\text{entry}}(s) \). Specifically, the sentry \( s \) is removed from any of the sets \( \text{Achievers}, \text{Invalidators}, \text{Guards}, \text{Terminators} \) in \( V(A) \). Note: if one wants to have a sentry visible but not invocable in \( V(A) \), an ‘execution’ constraint that is part of the ‘CRUDAE’ constraint specified below should be used.
- The set of stages of \( V(A) \) is a subset of \( \text{Stages}(A) \) excluding all stages, \( \{s\} \), that satisfy one of the following conditions:
  - \( s \) is included in an operation \( \text{hide}_{\text{stage}}(s) \) in \( C \) such that for any three stages in \( \text{Stages}(A) \), \( \{s, s', s''\}, C | s \in \text{Substages}(s'), s' \in \text{Substages}(s''), \text{hide}_{\text{stage}}(s') \in C \rightarrow \{s \mid s \in \text{Substages}'' \} \in V(A) \), i.e., the descendants of a hidden stage become direct descendents of its parent in the view schema, or
  - \( s \) is the descendant (i.e., \( s \in \text{Substages}(s') \)) and \( s' \) is included in an operation \( \text{condense}_{\text{stage}}(s') \) in \( C \).
- The set of milestones of \( V(A) \) is \( \{m | m \in \text{Milestones}(A) \} \) except if \( m \) satisfies one of the following conditions:
  - \( \text{hide}_{\text{milestone}}(m) \) \( \in C \).
  - \( m \in \text{Owns}(s') \) and \( s' \in \text{Substages}(s) \) and \( \text{condense}_{\text{stage}}(s) \in C \) (note: free-standing milestones should be explicitly included in \( \text{hide}_{\text{milestone}}(m) \) operation if need to be omitted in the view schema).

We provide some simple illustrations in connection with the running example and the GSM lifecycle schema of Fig. 2.

As with the FSM case, details concerning export restrictions can be hidden from the Customer participants. However, the Customer may have reason to see the actual export documents produced in the stage Prepare Export Documents, and have interest in knowing when they have been prepared. One way to achieve this is to condensing the stage Legal Reviewing, i.e., hiding everything inside it. For the Legal Services participants, both stages Requirements Gathering and Engineering Design can be eliminated. The stage Prepare Export Documents will rely on the milestone Design Approved (which would still be exposed to the Legal Services participant) and also attributes relating to the product design.

2) \( \text{Window}^{\text{GSM}} \):

A GSM-specific window relationship is defined in the same way as FSM-specific window.

3) \( \text{CRUDAE}^{\text{GSM}} \):

A GSM-specific CRUDAE relationship is defined in a similar way as FSM-specific CRUDAE relationship between a specific artifact type \( A \) and a specific participant \( p \). This relationship is associated with set of participant specific access rights, specified as mappings between \( \{R', U', A'\} \) actions and attributes in the information schema, between \( \{C', D'\} \) actions and the artifact type \( A \), and between the execution action \( \{E\} \) and sentries in the lifecycle of the corresponding artifact type \( A \).

A simple CRUDAE specification \( \alpha \) for \( A \) is a mapping with domain \( \bigcup_i \text{attr}(R_i) \cup \{E\} \) where:

- \( \alpha : (\bigcup_i \text{attr}(R_i)) \rightarrow 2^{R'^{U'}, A'} \)
- \( \alpha : (A) \rightarrow 2^{C', D'} \), to indicate participant’s privileges to create and delete instances of type \( A \), and
- \( \alpha(E) \subseteq Sentries \) (i.e., the subset of executable sentries in \( L_{\text{cycle}}(A) \)).

Aside from its design, a significant effort in ACSI is devoted to the actual implementation of both authorization models into a running prototype. This effort is described in the next section.

VI. IMPLEMENTATION

The implications of all the above restrictions being imposed to an I-Hub system imply the need to derive, and carefully determine execution eligibility of participant interactions, each being considered as having a distinct partial-view of the system.

To implement a multi-participant authorization model that can accommodate for all authorization restrictions mentioned above, a concrete object-model has been designed. This model is illustrated using UML class-diagram in Fig. 4 such that each model construct is represented as a class.
The meaning of each object model's construct is as follows:

- **ApplicationAuthorizationTransformationModel**: designates the root of the model containing authorization definitions for a single artifact-based application (i.e., identified by appName attribute). Each transformation model contains several authorization models, one for each artifact type in the application.
- **ArtifactAuthorizationTransformationModel**: designates authorization constraints for a single artifact type (identified by artifactId attribute). Each such constraint is a containment of several Views (i.e., ViewTransformationModel). Since a multiple ViewTransformationModels may be applicable to the current user, the CombiningAlgorithm construct specifies how multiple authorization views should be combined.
- **ParticipantFilter**: designates a condition on user information, determining the applicable authorization views to the current user.
- **InformationTransformation**: specifies which artifact instance data can be accessed. This includes:
  a) **SelectionWindow**: designates a selection function in the authorization model. The selectCondition function expression determines which artifact instances should become accessible. This condition may relate to artifact instance data or to current user information.
  b) **ProjectedAttributes**: designates a projection function in the authorization model. Attributes are uniquely identified by an xpath expression. Attributes that are not specified are not accessible. For each projected-attribute, two conditions may be specified: readCondition and writeCondition to determine whether a user can read/write values from/to an attribute.
- **LifecycleTransformation**: Abstractly specifies how the artifact lifecycle schema is exposed to a participant. This construct has two concretizations for FSM and GSM lifecycle schema styles that are listed below. The CreatePermitted and DeleteCondition slots determine whether the current user is legible to create or delete artifact instances.
- **FSM Transformation**: an FSM lifecycle transformation is composed of a set of CondensedStates and TransitionAccessControls:
  a) **CondensedState**: designates a condensation function in the authorization model. The name of the condensed state is specified in the name attribute, and all original states to be combined are specified in the originalStates collection attribute. Excluding certain circumstances, all original states and their outgoing transitions become hidden from the participant. An original state can participate at most in one condensation of a particular view.
  b) **TransitionAccessControl**: determines whether a transition can be executed by the participant. The ref attribute references a transition in the artifact's lifecycle schema. The executeCondition attribute then specifies a
corresponding invocation condition. When the condition holds the participant can invoke (execute) the referenced transition. Each original transition can appear at most once in the TransitionAccessControl collection.

- GSM Transformation: a GSM lifecycle transformation is composed of a set of ExposedSentries of type SentryAccessControl and AbstractionOperators:
  a) SentryAccessControl: designates authorization constraints for a referenced sentry (ref). The ExecuteCondition attribute specifies a corresponding invocation condition. When the condition holds the participant can invoke (execute) the referenced transition.
  b) AbstractionOperator: an abstract construct, designated to abstract stages or milestones from the underlying GSM schema. We distinguish among the following concrete operator types:
    - CondenseStage: Removes the substages (and their corresponding milestones) of the referenced stage from the GSM schema.
    - HideMilestone: Removes the referenced milestone from the GSM schema.

REFERENCES


- HideStageStatus: Removes the referenced stage status variable from the GSM schema. Note, that the milestones owned by the stage and the substages remain in the schema.
- MacroOperator: an abstract class allowing assembling composite operators.

VII. CONCLUSIONS

The approach to access control described in this paper is being implemented as part of the ACSI project. For the FSM approach to specifying lifecycles, a pilot implementation has been created and demonstrated on top of the Siena engine [7]. In particular, we have extended Siena with the capability to both specify and interpret an XML description (i.e., instance of the above object-model), created on a per application level. This description is used at runtime to enforce all relevant access-control constraints. Hence, this recent release supports the interpretation of FSM-based authorization restrictions. The team is currently immersed in the implementation of similar functionality in Barcelona [11] to support the access controls in the GSM context. This effort is expected to complete by the end of 2012.