The core ACSI interoperation hub framework
Abstract

This report describes the Interoperation Hub (aka the I-Hub) abstract model that underlies the ACSI hub framework. This model extends the core ACSI Artifact Abstract Model (A³M) described in D1.1 to support multiple-enterprise artifact-centric systems focused primarily on facilitating communication and business-level synchronization between relatively autonomous stakeholders (and stakeholder organizations).

I-Hubs provide a centralized, computerized rendezvous point where stakeholders can read or write data of common interest. They can also check the current status of an aggregate process, from which they can receive notifications about events of interest. At its core, an I-Hub is designed using “business artifacts”, a data-centric paradigm for workflow and business process specification. This report details how the core artifact-based system is extended to support multi-participant capabilities.

The report starts with a broad definition of an I-Hub abstract model, specifying all the features required to enrich it for multi-user configurations. The model is then further elaborated for concrete artifact systems, whose artifacts’ lifecycles are described as finite state machines. We conclude this report with a full architecture and realization description of the developed prototype that is designed to demonstrate the actual feasibility of the capabilities theorized.
Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1.0</td>
<td>23-02-2011</td>
<td>First draft</td>
</tr>
<tr>
<td>V2.0</td>
<td>03-03-2011</td>
<td>Post F2F meeting in Tallinn</td>
</tr>
<tr>
<td>V3.0</td>
<td>22-05-2011</td>
<td>Order-to-Cash example adjustments</td>
</tr>
<tr>
<td>V4.0</td>
<td>23-05-2011</td>
<td>Service realization adjustments</td>
</tr>
<tr>
<td>V5.0</td>
<td>30-05-2011</td>
<td>Submitted</td>
</tr>
</tbody>
</table>
| V6.0    | 31-10-2011| Re-submission based on reviewer’s suggestions in the annual review:  
- Typo corrected on p14 (“lifestyle’’). We also wish to note that we consider the correspondence between the two lifecycle styles GSM and FSM such that the former subsumes the latter.  
- Partner on-boarding will be handled in the specification of variation-points and customisation. This will be specified in detail in D1.4.  
- An example was added on p14, clarifying the notion of ‘interoperability’.  
- ACSI time acknowledges that further work has to be carried out to align between ACSI’s vocabulary and the one in WSDL 2.0. A corresponding committee has been set up for this purpose within the ACSI team. Results will be reported in D1.4. |
Table of Contents

Abstract ......................................................................................................................................................... 2
Document History ........................................................................................................................................ 3
Table of Contents ...................................................................................................................................... 4
List of Figures ............................................................................................................................................... 5
Acronyms...................................................................................................................................................... 6
1. Core Interoperation Hub Framework ................................................................................................. 7
   1.1. An Abstract Basis for I-Hub Models .............................................................................................. 8
   1.2. Concrete Models .......................................................................................................................... 15
       2.1.1. FSM – Further Refinement of I-Hub specific features .................................................. 15
       2.1.2. FSM – Order-to-Cash Example – Instantiation of I-Hub Specific Features ............ 17
       2.1.3. GSM – Further Refinement of I-Hub Specific Features ................................................ 20
       2.1.4. GSM - Requisition Order Example – Instantiation of I-Hub Specific Features .... 20
2. ACSI I-Hub Realization ....................................................................................................................... 20
   2.1. The ACSI-Hub Use Cases .............................................................................................................. 21
   2.2. The ACSI-Hub Authorization View Model .................................................................................. 22
   2.3. I-Hub Main Components ............................................................................................................. 26
   2.4. I-Hub Internal Package Structure ................................................................................................. 28
   2.5. The ACSI-Hub Framework – Runtime Demo ................................................................................ 30
3. Appendix 1 – Example for Service Type Interface Description ....................................................... 31
List of Figures

Figure 1 - Core I-Hub Framework ........................................................................................................ 8
Figure 2 - Service binding mechanism ................................................................................................. 11
Figure 3 - Condensation example ......................................................................................................... 20
Figure 4 - I-Hub prototype system use cases. A typical sequence of interactions is clockwise .. 21
Figure 5 - I-Hub Authorization Model class diagram ........................................................................... 23
Figure 6 - Authorization View XML example ........................................................................................ 26
Figure 7 - I-Hub main components ....................................................................................................... 26
Figure 8 - Packages in the I-Hub component. Arrows represent a dependency relationship between packages .................................................................................................................................................. 29
Figure 9 - Manufacturer/Supplier views ............................................................................................... 31
Figure 10 - Customer/Manufacturer views ............................................................................................ 31
## Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACSI</td>
<td>Artifact-Centric Service Interoperation</td>
</tr>
<tr>
<td>I-Hub</td>
<td>Interoperation Hub</td>
</tr>
</tbody>
</table>
1. Core Interoperation Hub Framework

The ACSI interoperation hub framework develops the concept of an Interoperation Hub (aka the I-Hub) along several dimensions, so that it can become an enabler of open, scalable, long-tail service interoperation.

This work is centered around creating a comprehensive framework for interoperation hubs and participating services, including (i) developing the capacity for specifying and enforcing sophisticated access controls for different participating services, (ii) extending the interoperation hub concept in literature to support collaboration environments, rather than single-service collaborations, and (iii) developing techniques that enable the hub engine to support data and tasks in both virtual and material ways. The ACSI interoperation hub framework will be designed to take full advantage of the richness of the ACSI artifact paradigm, and participants can select increasingly sophisticated interoperation modalities. The goal is to simplify the job of both service collaboration designers and the users of participating services, enabling them to participate in, customise, or substantially re-design service collaborations. We foresee three essential forms of interoperation, each with increasing flexibility: (a) Use-only interoperation: Making use of one or more artifact types provided by the interoperation hub; (b) Customise-only interoperation: Customising one or more artifact types in the interoperation hub, thus introducing new, but derived, artifact types in the hub. Such customisations essentially amount to using a subset of the full capabilities of the artifact type. (c) Full interoperation: Creating new artifact types in the interoperation hub. The increasing level of flexibility obviously corresponds to an increasing level of reasoning support required by the interoperation hub in setting up the new artifacts and managing them at runtime.

At this stage of the project, within WP1 we are focusing on introducing an abstract basis for interoperation hubs, which extend the ACSI Artifact Abstract Model (i.e., the A3M model) to address the notion of different participating stakeholders. The core Interoperation Hub (I-Hub) framework (see Figure 1) is a combination of: (1) an implementation-agnostic model that presents the very basic principles that underlie an artifact system (i.e., the A3M model); (2) an abstract I-Hub model that extends the A3M model with all I-Hub specific features and the required conceptualizations to address multiple-participant service-interoperations (e.g., participant authorizations); (3) concrete models that illustrate possible lifecycle styles for the realization of I-Hub systems; and (4) implementations that correspond to each conceptual model. The focus of the next section is the Abstract I-Hub Model. Hence, this model is consistent with all the definitions in the ACSI Artifact Abstract Model (A3M).
1.1. An Abstract Basis for I-Hub Models

Essentially, an interoperation hub system is a specialization to an artifact system, extending it with additional hub-specific features to handle multi-participant collaborations. As described in the A³M model (D1.1), an artifact-centric system interacts with the environment through gateways. For example, participants’ access to an artifact and its environmental components (e.g., external services) is achieved through the data stored in certain gateways that are akin to the artifact. However, an artifact-centric system is basically agnostic concerning the concrete existence of participants and related properties in the environment, which are needed to support the facilitation of service collaborations. This may include properties such as the identity of participants, their organisational affiliations, roles, assigned access privileges, and awareness of the underlying semantics of the information in the artifacts. Such information is essential in an I-Hub system to properly manage multi-participant interactions. Therefore, in the generic conceptual I-Hub model we extend the A³M model as follows.

**I-Hub system**: An I-Hub system is a tuple $H = (M, Par, Ser_p, R^H, C^H, D^H)$, where:

- $M$ is an artifact system (i.e., specified in the A³M model).
- $Par$ is a set of all participant-specific properties detailed in the next section.
- $Ser_p$ is a set of the provided services by an I-Hub. Note: The notion of a provided service is meaningful only to elements (e.g., participants) in the environment. Invocation of such services by participants entails the transformation of incoming ‘messages’ into corresponding events inside the artifact system. Such events may affect one or more artifacts in $M$. This transformation of messages to events is accounted by a service binding mechanism (described below).
- $R^H$ is a set of hub-specific relationship types. Note: This is a disjoint set from the set relationship types $R$ in $M$, such that: $R \cap R^H = \emptyset$. In the I-Hub, the set of relationship
types extends the basic set of relationship types in an artifact system $M$ to account for the following:

1. Denote links that reflect participant-wise organisational structures (e.g., employee roles).
2. Prescribe event-to-service and service-to-event linkages to the service binding mechanism.
3. Specify constraints over possible interactions between participants and elements in the artifact system (namely, *authorization views*).

Major relationship types for each of the above purposes are further detailed below.

- $C^H$ and $D^H$ are, respectively, a set of static and dynamic constraints that characterize the mechanics of hub participant-specific properties and relationship types. These constraints can be specified in terms of logical formalisms, similarly to what shown in D1.1. Here, however we describe these mechanisms in a semi-formal way.

The set $Par$ in a hub system is used to specify the various elements associated with each individual participant. This mainly includes organisations, role affiliations, and services that the I-Hub may require from each participant (i.e., consumed services). Thus, in an I-Hub system we define the set $Par$ as a set of participant-specific properties, such that each participant $p \in Par$ is defined as a tuple $p = (Org, Rol, Ind, Ser_r)$, where:

- $Org$ – a set of organisational affiliations associated with an individual participant. Typically, this will include the identification of a single organisational unit. However, in its most generic form, a participant may also be associated with a consortium of several organisational units, to be considered as a single silo.
- $Rol$ - a set of role types. The humans eligible to perform tasks or send messages can be grouped into roles. Such roles can be further divided into *static roles* to denote the different classes of employment in each organisation (e.g., research staff member) and *contextual roles* to denote artifact-to-participant interaction specific roles (e.g., the author of a research paper).
- $Ind$ – a set of individuals (i.e., I-Hub application users) who actually interact with the I-Hub on behalf of their appointed organisational roles. Through the specification of respective relationship type in $R^H$ between individual and elements in the artifact system, the I-Hub can restrict who can perform what at a given time according to role names (i.e., static or contextual) and organisational affiliations (see relationship types below).
- $Ser_r$ – a set of (one-way or two-way) participant-specific services that can be consumed by the I-Hub (i.e., provided to the I-Hub by participants in the environment).

Both *provided* (i.e., $Ser_p$) and *consumed* (i.e., $Ser_c$) services in an I-Hub system have a type. A *Service type* is used to specify the characteristics of a family of service executions. It includes the name of the service type, the payload schema of message calls for the service, and the payload schema of message responses by the service. The attributes of these payloads are also
associated with a concrete set of gateway fields in the A³M system, and may be indicated as mandatory or optional. More concisely, the specification of each service type follows the following formalism:

**Service type**: the full specification of a service type is a tuple

\[ ST = (s, If, ifsid, endpoint, Pre, Mifs) \]

where:

- \( s \) is a service type name.
- \( If \) is an interface specification URI of the service implementation. This specification can be used to retrieve the full description of the service’s payload schema.
- \( ifsid \) is an identifier of the service in the interface specification.
- \( endpoint \) is a service endpoint URI.
- \( Pre \) is a set of preconditions, which must be satisfied to apply the service (e.g., the user must have used a login service to get the security token).
- \( Mifs \) is a mapping from a service implementation interface to the corresponding gateway.

In an I-Hub system, we further elaborate the **service binding mechanism** that handles interactions between participants and artifacts in the artifact system. Unlike internal interactions between artifacts within the artifact system \( M \), which are accomplished strictly through the management of events (described in the A³M model), interactions between artifacts and external participants is facilitated through the exchange of messages (incoming or outgoing). A **service binding mechanism** modulates between an internal stream of events and an external stream of messages.

As illustrated in Figure 2, the binding mechanism has two responsibilities:

1. For each outgoing event that requires the invocation of a consumed service, the binding mechanism will construct one or more corresponding outgoing message(s) according to the specification of the related consumed service type.
2. For each incoming message constructed by an external participant according to a certain provided service type specification, the binding mechanism will construct one or more corresponding incoming event(s) to be consumed internally by artifacts.

Concrete I-Hub implementation architectures may bypass the explicit implementation of a service binding mechanism by delegating its responsibility across the internal artifacts that may themselves be responsible in such cases for the construction of outgoing messages and the interpretation of incoming messages.
Note, nothing is stated in the I-Hub model about the actual density of services provided to participants (i.e., portion of internal events exposed to external participants) or about the mappings to the artifact schema. This must be specified for each concrete model (e.g., FSM, GSM). However, to ensure that all concrete models adhere to the notion of an I-Hub as a structured white-board that participants can update and refer to with their related information, each concrete model should define its service density and mappings in a structure that adheres to the following principle:

**Service faithfulness:** For each participant, the services provided by the hub and their internal bindings should expose an interface that is isomorphic to all the internal lifecycle “phases” that are accessible (i.e., not restricted by any authorisation view) to that participant. In the most permisive settings (i.e., no authorisation restrictions), the exposed interface includes all the internal “phases”. This principle also reflects a commitment in an I-Hub system to ensure that the detail of its internal computation is hidden, exposing a very clean artifact-based conceptual view to the outside world.

The set $R^H$ of relationship types in a I-Hub system, with associated constraints in $C^H$ and $D^H$, relates elements in $H$ internally and also with elements in $M$. Specifically, this includes the following relationship types:

- **Organisational structures** – possible links between elements in $Par$ to denote organisational structures (e.g., between individuals and roles). Examples of such relationship types are the following:
  - **Org-to-Role:** designates for each organisational affiliation its set of corresponding static roles.
o **Ind-to-Role**: designates for each individual a role (one-or-more) to which the individual has been appointed.

- **Service bindings** – establish the information on which the service binding mechanism described above is reliant upon to determine the mappings between messages and events in $M$. Depending on message direction (i.e., incoming or outgoing), its destination may be either an element in $S_{r_p}$ or in $S_{r_c}$. Therefore, to fully specify the mapping between events and messages, the following relationship types are defined:
  
  o **EventType-to-$S_{r_c}$**: specifies for each outgoing event type, one or more participant-specific services to be invoked (i.e., through the construction of corresponding messages).
  
  o **$S_{r_c}$-to-EventType**: specifies for each provided service invocation (i.e., identified as an incoming message), one or more incoming event types to be constructed.

- **Authorisation views** – these relationship types are between participant elements in $Par$ (e.g., individual users) and elements they may wish to access in $M$ (e.g., artifact information attributes). Each such relationship type is associated with constraints in $C^H$ and $D^H$ to specify the various access restrictions for the interaction of each participant with the I-Hub system (e.g., runtime access controls). This set includes the following (non-exhaustive) relationship types:
  
  o **View (par-to-$\text{artifactType}$)**: a view relationship type describes a connection between each participant $p$ in $Par$ (e.g., an individual or a role) and a specific artifact type $A$ in $M$. Each such relationship type is then associated with a set of constraints in $C^H$ and $D^H$ to restrict what parts of an artifact an actual user can see. With regards to artifact types, the notion of view includes two restriction types:
    
    - Restrictions on the attributes that can be seen.
    - Restrictions on the portion of the lifecycle phases that can be seen.

  In terms of the latter, the set of view constraints exposes a participant-specific transition system $p^M$ that simulates the original system $M$ to participant $p$. That is, for each phase transition in the original lifecycle model $M: I \xrightarrow{\varphi} I'$ that is specified by a constraint $\varphi$ (in a certain language, e.g., LTL), there is a derived constraint $\hat{\varphi} = p(\varphi)$ (e.g., in language ATL) specifying the legibility of the corresponding transition $p: I \xrightarrow{\hat{\varphi}} I'$ in the participant’s view. It is important to note that for each transition in $M$ there must be at least one participant who can execute it. Also, the transition is executed as a whole by each legible participant.

  o **Window (par-to-$\text{artifactType}$)**: a window relationship type describes a connection between a participant $p$ in $Par$ (e.g., an individual or a role) and a specific artifact type $A$ in $M$. Each such relationship is then associated with a set of constraints in $C^H$ and $D^H$ to restrict which artifact instances $a_1 \ldots a_n \in A^I$ of type $A$ an actual user can access.

  o **CRUDAE ($\text{artifactTypePhase-to-par}$)**: is a relationship between a specific phase $s \in S$ in the lifecycle of an artifact type $A$ and a specific participant $p$ in $Par$ (e.g., an individual or a role). This relationship is associated with a set of participant-
specific access rights, specified as mappings between 
{‘Create’,‘Read’,‘Update’,‘Delete’,‘Append’} actions and attributes in the
information model, and between the execution action {‘Execute’} and phase-
traversing steps in the lifecycle of the corresponding artifact type.

A **snapshot** of the I-Hub system $H$ is a function $J'$ that associates:

- to $M$, the snapshot of $M$ (as illustrated in D1.1)
- to each participant in $p \in Par$, the following:
  - to $p$, a unique participant identifier $PID$
  - to $Org$, a finite set of organization $OIDs$ (typically one)
  - to $Role$, a finite set of role $RIDs$ (typically role names may be used)
  - to $Ind$, a finite set of individual $IDs$ (e.g., employee IDs)
  - to $ServC$, a finite set of consumed service $SID$s, each of which is also related to
    a specific service type specification using the relationship type $ServC$-to-
    ServiceType in $R^H$.
- to $ServP$ a finite set of provided service $SID$s, each of which is also associated to a
  specific service type specification using the relationship type $ServP$-to-ServiceType in
  $R^H$.
- in addition to the above relationships and the ones already specified in the A3M
  model, the snapshot function also associates to each hub-specific relationship in $R^H$,
  the following:
  - Organisational structures:
    - to each $Org$-to-$Role$ relationship, $\rho^{org\rightarrow role}: OID \rightarrow RID$ is the
      organisation-to-role mapping (one-to-many).
    - to each $ind$-to-$Role$ relationship, $\rho^{par\rightarrow role}: ID \rightarrow RID$ is the
      mapping of role responsibilities to individual employees (one-to-
      many).
  - Service bindings:
    - to each $EventType$-to-$ServC$ relationship, $\rho^{eventType\rightarrow ServC}: EventType \rightarrow SID$s
      is the (one-to-many) mapping between event types in $M$ and consumed services.
    - to each $ServP$-to-$EventType$ relationship, $\rho^{ServP\rightarrow EventType}: SID \rightarrow
      EventType$ is the (one-to-many) mapping between provided services and event types in $M$.
  - Authorisation views:
    - to each $View$ relationship type, $\rho^{view}: PID \times A \rightarrow red(a')$ is the
      participant-to-artifact type mapping to which a specific “reduction”
      function $\text{red}$ is used to filter out some portion of the database
      instance $a'$ for schema $D_A$, such that only a subset of the attributes
      and a single lifecycle phase is included.
to each Window relationship type, $I^{\text{window}}: PIDs \times A \rightarrow AP$ is the participant-to-artifact type mapping to which a (sub) set $AP \in P(A^I)$ of artifact instances are specified as accessible.

- to each CRUDAE relationship type, $I^{\text{CRUDAE}}: PIDs \times S \rightarrow crudae(A)$ is the participant-to-phase mapping to which the access-right function $crudae$ then assigns mappings between \{'C','R','U','D','A'\} actions and each attribute in the information model of $A$, and between the execution action \{'E'\} and a set of permitted phase transitions $(I,I') \in F$ in the lifecycle of $A$.

Based on the above definitions, we now define the **criterion for interoperability**: 

Let an artifact system $M$ be considered a graph $G = (A, R_A)$, such that artifact types $A$ are considered as vertices and artifact relationship stereotypes among artifacts $R_A$ are considered as edges. An interoperability criterion is met if there exist at least one connected sub-graph in $G$ for which the number of distinct participants in the set $Par$ is greater than one. This entails two possible types of interaction: direct and indirect, as illustrated in Figure 1. Direct interoperability exists when two (or more) participants interact by affecting (and being affected by changes in) the lifecycle of a mutual artifact instance as in the case of an author and a reviewer that interact on the basis of a mutual conference paper. Indirect interoperability exists when two (or more) participants interact by affecting (and being affected by changes in) the lifecycle of two or more artifact instances, each which may affect the lifecycle of the other. As illustrated, an example may be a designer being responsible for the specification of a requirement while indirectly the same requirement’s instance could affect the state of a concrete work-task being managed by a corresponding software developer.

![Figure 3 - Interoperability](image)

**Figure 3 - Interoperability**

An I-Hub system is an artifact system for which the interoperability criterion is met.
1.2. Concrete Models

2.1.1. FSM – Further Refinement of I-Hub specific features

In this section, we further specify the features of an I-Hub system so that it fully complies with a system of artifacts whose lifecycles are described by explicit finite state machines (FSM). The next section uses the Order-to-Cash example to demonstrate how each of the I-Hub specific features can be used to specify an actual application that involves service interoperability between multiple stakeholders.

All the I-Hub features described above are formulated to make them applicable to any artifact lifecycle specification, including an artifact system with lifecycles illustrated as FSMs. In this section, we further refine some of the definitions to clarify the nature of the I-Hub’s specific features for an artifact system with artifact lifecycles specified as FSMs. This will assist with the specification of applications, such as the one presented in the example below. The features not further refined in this section should be defined as outlined in Section 1.1.

Formally, in FSM-specific specifications, we refer to a specialized artifact system $M^F$ whose artifact lifecycle is described by explicit finite state machines (FSMs), such that the lifecycle of an artifact type $A$ is specified by an artifact lifecycle schema $L = (S, E)$, 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 $S$), aka ”transitions”, such that there are no in-edges into source and no out-edges from sink.

To refine the specification, we adjust the generic terminology used to describe the dynamics of an artifact system with specific constructs that appear as first class residents in an FSM model. This essentially includes two major concept refinements: the "state" construct in FSM replaces the somewhat abstract "phase" notion, and the "transition" construct in FSM replaces the notion of a "step" (i.e., phase traversing) in a general artifact system. Based on these two conceptualization refinements, we can further elaborate the service faithfulness principle for an FSM-based I-Hub as follows.

**FSM-specific service faithfulness**: for each participant, the services provided by the hub and their internal bindings should expose an interface that is isomorphic to all internal lifecycle transitions that are accessible (i.e., not restricted by any authorization view) to that participant. In the most permissive settings, the exposed interface includes all the internal transitions.

Other than the service faithfulness principle, a significant refinement is made to the specification of authorization views. This enables a more concise definition of the mechanisms that ensure that information and events that should be kept private and invisible from some participants are indeed being kept as such when artifacts lifecycle are FSMs. Specifically, we further specify the following relationship types:
- **View**: an FSM-specific view relationship relates between a participant $p$ in $\text{Par}$ and a specific artifact type $A$ in $\mathcal{M}^p$. Each such relationship type is then associated with a set of constraints in $C^H$ and $D^H$ to restrict what parts of an artifact an actual user can see.

  This includes:
  - Restrictions on the attributes that can be seen, achieved using attribute "projection":
    
    Let $a_1 \ldots a_n = \bigcup_i \text{attr}(R_i)$ 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 $a_1 \ldots a_n$.
  - Restrictions on the set of states that can be seen, achieved using "node condensation":
    
    A condensation mapping over a lifecycle schema $L = (S,E)$ is an expression $\gamma_f$ where $f : S \rightarrow S'$ is a surjective function over $S$, such that:
    
    - $\text{source, sink} \in S'$.
    - $\gamma_f(\text{source}) = \text{source}$ and $\gamma_f(\text{sink}) = \text{sink}$.
    - $\gamma_f(L) = (S',E')$ where $E' = \{(\gamma_f(s_1),\gamma_f(s_2)) | (s_1,s_2) \in E\}$.

  Note: multiple states of $S$ can map into $\text{source}$ or $\text{sink}$.

As described in the generic form of a view in section 1.1 above, the specification of node condensations segregates the original lifecycle model into several participant-specific transition systems weakly simulating the original lifecycle behavior with distinct transition execution responsibilities\(^1\). Specifically in FSM lifecycles, it is important to single out an implied principle with respect to each condensed state in $S'$ and participant-specific transition system $p^M$. We call this principle the principle of execution disambiguity and define it as follows:

- Participant $p$ can execute a transition from any condensed state $s_i \in S'$ to any other state $s_j \in S'$, either condensed or not, iff all original condensed states $s_o \in S$ jointly condensed to $\gamma_f(s_o) = s_i$ are also linked to the same target state $s_j \in S$ in the original transition system, such that $E' = \{(\gamma_f(s_o),\gamma_f(s_j)) | (s_i,s_j) \in E\}$.

  Simply put, since state condensation results in hiding the internal original states from certain participants, exiting from a condensed state is only possible if all original states that were jointly condensed include a transition to the same target state that is not within the original condensation. An example for such a case would be a participant’s request to ‘cancel’ the process while it is in a condensed

\(^1\) We acknowledge there is a place for further formal investigation on how these transition systems are related to each other and also to the original one. This will be studied further in ACSI.
state. Note: The mutual target state in such a case could also be part of another condensation.

- **Window**: an FSM-specific window relationship relates between a participant \( p \) in \( Par \) and a specific artifact type \( A \) in \( M \). Each such relationship is then associated with a set of constraints in \( C^H \) to restrict which artifact instances of type \( A \) a participant can access. This is achieved using instance ‘selection’:

  Let \( \sigma_\varphi(A): A^I \rightarrow A^I_p \) be a unary operation over artifact type \( A \), \( \varphi \) is a propositional formula using the union set of attributes \( a_1 \ldots a_n = \bigcup_i attr(R_i) \) of all relations in \( D_A \), such that:

  - \( A^I_p \) is a subset of \( A^I \) of all artifact instances legible to be seen by participant \( p \).
  - \( A \sigma_\varphi(A) \) restriction is defined for all lifecycle states in \( L(A) \).

Note: The same selection constraint may be applied to all lifecycle states.

- **CRUDAE**: an FSM-specific CRUDAE relationship between a specific state \( s \) in the lifecycle of an artifact type \( A \) and a specific participant \( p \) in \( Par \). This relationship is associated with set of participant specific access rights, specified as mappings between \( \{C',R',U',D',A'\} \) actions and attributes in the information model and between the execution action \( \{E'\} \) and transitions in the lifecycle of the corresponding artifact type. A simple CRUDAE specification \( \alpha \) for \( A \) is a mapping with domain \( \bigcup_i attr(R_i) \cup \{E'\} \) where:

  - \( \alpha: (\bigcup_i attr(R_i)) \rightarrow 2^{\{C',R',U',D',A'\}} \)
  - A possible simplification for ‘create-delete’ actions may be applied on an artifact type level: \( \alpha: (A) \rightarrow 2^{\{C',D'\}} \)
  - \( \alpha(\{E'\}) \subseteq E \) (i.e., the subset of executable transitions in \( L(A) \)).

### 2.1.2. FSM – Order-to-Cash Example – Instantiation of I-Hub Specific Features

This section illustrates how the Order-to-Cash example can be instantiated to fully specify a concrete application that uses the features of an I-Hub system.

The instantiation adheres to a system of artifacts whose lifecycles are described by FSMs. Specifically, we instantiate an I-Hub system \( H = (M, Par, Ser_p, R^H, C^H, D^H) \), such that each of its comprising elements is instantiated as follows:

- \( M \) is an artifact system according to \( A^M \) and comprises four artifact types,
  - \( A_1 = Customer\ purchase\ order\ (CPO) \)
  - \( A_2 = Work\ order\ (WO) \)
  - \( A_3 = Material\ purchase\ order\ (MPO) \)
  - \( A_4 = Line\ item\ (LI) \)
- \( Par = \{(Org, Rol, Ind, Ser_c)\} = \)
  - \( p_1 = \{ ([AT&T], \{customer\}, \{John, Barbara\}, \Phi) \}
  - \( p_2 = \{ [Apple], \{manufacturer, procurementOfficer\}, \{Steve, Smith\}, \Phi \)
In other words, this is an example for a specification of four distinct participating organizations: a customer, a manufacturer, and two suppliers. Each participant’s specification is instantiated as a set of organization names, roles, individual users, and empty sets of consumed services. (In this example, the I-Hub does not use the feature of consumed service).

- \( Ser_p = \)
  - \( \{ (\text{addLineItemToMPO}) , \)
  - \( (\text{sendMPOToSupplier}), ... \) \)

This is an example for the specification of two service types provided by the hub to its environment, each with its corresponding operation signature.

An example for the specification of a concrete service type, including its reference to a certain URI descriptor for the service’s payload schema (e.g., wsdl), is as follows:


Please refer to Appendix 1 for an example of a service descriptor that details in full the input and output payload schemas of a service type.

- \( R^H = \)
  - Organizational structure specifications:
    - \( Org - to - Roles = (\text{Intel}, \{\text{supplier}, ...\}), ... \)
    - \( Ind - to - Roles = (\text{John}, \{\text{customer}, ...\}), ... \)
  - Service binding specifications:
    - \( Ser_p - to - EventType = \)
      - \( (\text{addLineItemToMPO}, \{\text{MPO AddItem}\}), ...) \)
(sendMPOToSupplier, \{MPOSendToSupplier \*\}),

* followed by a sequence of internal events for each line item on LineItem.

- **EventType** = to - Ser_c

* no event-to-service instantiations since no consumed services have been defined in this example.

- **Authorized view:**
  The following as an example of how the components of an authorization view could be specified for the role of customer and the artifact type CustomerPurchaseOrder:

  - **View:**
    - project(CPOID, productCode, customerID, manufacturerID, status) excluding the attribute WOID from the customer's view.
    - condense(WO created, MPOs sent, MPOs received) such that:
      - \( \gamma_f(WO \text{ created}) = \text{inProgress} \)
      - \( \gamma_f(MPOs \text{ sent}) = \text{inProgress} \)
      - \( \gamma_f(MPOs \text{ received}) = \text{inProgress} \)
    
    The original transition system and the resulting customer's view are illustrated in Figure 4.

  - **Window:**
    - \( \varphi = (\text{customerID} = $\{user\}.ID) \), to ensure the current logged users can only see artifact instances they themselves created.
    - Apply \( \varphi \) to all lifecycle states.

  - **CRUDA:**
    - \( \alpha: (\text{CustomerPurchaseOrder}) = \{C', D'\} \), the customer can create and delete artifact instances. Furthermore, apply the following access controls to each attribute:
      - \( \alpha: (\text{CPOID}) = \{R', U'\} \)
      - \( \alpha: (\text{customerID}) = \{R', U'\} \)
      - \( \alpha: (\text{productCode}) = \{R', U'\} \)
      - \( \alpha: (\text{manufacturerID}) = \{R', U'\} \)
      - \( \alpha: (\text{status}) = \{R', U'\} \)
    - \( \alpha: (E') = \{\text{sendToManufacturer, customerCancels}\} \),
    The customer can only execute the transitions listed.
2.1.3. GSM – Further Refinement of I-Hub Specific Features

According to the ACSI work plan, this part of work is in progress and will be reported in D1.4 - The complete ACSI Hub - deliverable.

2.1.4. GSM - Requisition Order Example – Instantiation of I-Hub Specific Features

According to the ACSI work plan, this part of work is in progress and will be reported in D1.4 - The complete ACSI Hub - deliverable.

2. ACSI I-Hub Realization

This part of the report illustrates the prototype system that is being developed as a realization of the I-Hub model. Specifically, the current prototype adheres to the I-Hub model whose artifact lifecycle is specified as FSMs. The focus of the current implementation has been on extending the basic Siena engine (i.e., an artifact-based system developed by IBM prior to ACSI that adheres to the A³M model) with the capability to control multi-participant collaborations. Thus, the focus in this section is on the complementary capabilities that have been added to enrich the given engine with all the I-Hub specific features explained in Section 1. The software package reflecting the current version of this prototype (i.e., ver 1.0) was submitted to the EC through the ACSI project management website as part of the D4.1 deliverable. An up-to-date “production” version of the prototype is also publicly accessible at the following link:
The latter is being continuously updated as the development effort in WP4 progresses.

2.1. The ACSI-Hub Use Cases

This section specifies the functional requirements from the I-Hub prototype implementation, focusing on typical interactions between external users and the system. This gives the context for possible system usage.

Figure 5 - I-Hub prototype system use cases. A typical sequence of interactions is clockwise.

Actors:

- **Application Designer**: responsible for designing and deploying core artifact-based applications and participant-specific access restrictions.
- **Administrator**: responsible for managing users (individual participants and organizations) and assigning roles.
- **Participant**: a certain stakeholder that uses an artifact-based application.

Use cases:

- **Design Application (A³M)**: definition of artifact-based applications, including: specification of artifacts, their information-model and lifecycle-model, and service bindings.
- **Design Authorization Views**: definition of participant-specific views. For each participant and artifact type, the designer may specify accessible portions of the artifacts' information-model and lifecycle.
- **Deploy Application**: embedding of all application specifications into an executable engine.
- **Membership Management**: definition of participating organizations and all corresponding information (e.g., roles, system users, passwords).
- **Authenticate Participant**: authentication of a participant’s identity (e.g., by password).
- **Application Model Inquiry**: participant-specific metadata retrieval. That is, legible portions of the artifact schema.
- **Application Execution**: running deployed applications and synchronizing the interactions between participants and artifact instances.

### 2.2. The ACSI-Hub Authorization View Model

To support the I-Hub required features, the Artifact Authorization Views model has been implemented. When creating an application, this model is augmenting the basic specification of the corresponding artifact-based application model that is defined in the Siena system. Hence, using UML class diagram notation, the model in Figure 6 illustrates the constructs (represented as classes) that have been created to realize the various aspects of authorization in an I-Hub application to control multi-participant interactions.
Figure 6 - I-Hub Authorization Model class diagram

The content of each model’s construct is as follows:

- **ApplicationAuthorizationTransformationModel**: the root of the model contains authorization definitions for a single artifact-based application (identified by `appName` attribute). Typically, each transformation model contains several `ArtifactAuthorizationTransformationModels`, one for each artifact type in the application.

- **ArtifactAuthorizationTransformerModel**: contains authorization constraints for a single artifact (identified by `artifactRefId` attribute). The authorization data is packaged as views (`ViewTransformationModel`). In this level of model hierarchy, each view’s specification is focused on a single artifact, identified by its `id` attribute (globally identified by `<appName, artifactRefId, id>`).
  - **ParticipantFilter**: specifies a condition on user information. At runtime, this condition determines which views are applicable to the current user. The specification of this condition is mandatory. Since a certain filter expression may result in having several `ViewTransformationModels` being applicable to
the current user, the CombiningAlgorithm construct should be used to specify how multiple authorization views should be aggregated if there are conflicting constraints (e.g., a denial constraint overrides a permission constraint or vice versa).

- **InformationTransformation**: specifies how the artifact instance data can be accessed by a participant. If this node specification is omitted, all artifact data is made accessible to the participant.
  - **SelectionWindow**: the `selectCondition` attribute expression determines which artifact instances should become accessible to a user. The condition may refer to the artifact instance data or to the current user information. The specification of a selection condition is optional. If omitted, the user can access all instances.
  - **ProjectedAttributes**: determines all information-model attributes that can be accessed by the user. The attributes are uniquely identified by an `xpath` expression. Attributes that are not specified are not accessible. For each projected-attribute, two conditions can be specified: `readCondition` and `writeCondition` determine whether a user can read/write values from/to the attribute. The specification of both conditions is optional. If omitted, the attribute is accessible both for reading/writing.

- **LifecycleTransformation**: specifies how the artifact lifecycle model is exposed to a participant. If this whole node specification is omitted, the complete lifecycle execution model is exposed to the user.
  - **CreatePermitted**: specifies a condition over participants’ information. At runtime, this condition determines whether the current user is authorized to create new artifact instances. If the condition is omitted, artifact creation is permitted.
  - **DeleteCondition**: specifies a condition over both participants and artifacts’ information. At runtime, this condition determines whether the current user is authorized to delete artifact instances. If the condition is omitted, artifact deletion is permitted.
  - **CondensedState**: specifies a set of states in the complete artifact’s lifecycle model that should be combined as a single condensed state visible to the participant. The name of the condensed state is specified in the `name` attribute (mandatory), and all original states to be combined are specified in the `originalStates` collection attribute. Typically, all original states and their outgoing transitions become hidden from the participant. (Note: For a more concise implication of the condensation constraint see Section 2.1.1 above). An original state can participate in zero or one condensation within a particular view.
  - **TransitionAccessControl**: determines whether a transition can be invoked by the participant. The `originalTransitionRef` references a transition in the artifact’s lifecycle model. The `executeCondition`
specifies the invocation condition. When the condition holds, the participant can invoke (execute) the referenced transition. If a condition specification is omitted, by default the transition is made executable to the participant. However, transitions that are not referenced cannot be executed. Each original transition can appear zero or one times in the TransitionAccessControl collection.

A concrete XML description that adheres to the structure of the illustrated model is created per application to specify all the authorization constraints to be enforced while the application is being executed and multiple participants interact with it. An example for a concrete XML file which adheres to the above model is illustrated in Listing 1. This XML specification applies several authorization constraints to any user who has been appointed the requester role when interacting with the PurchaseOrder artifact as part of the OrderToCash application.

```xml
<ApplicationAuthorizationModel
    xmlns="http://ihub.ibm.com/models/AuthorizationModel"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ihub.ibm.com/models/AuthorizationModel C:\Siena\GalileoWorkspace\acsi.iHub\resources\IHub.xsd"
    appName="OrderToCash">
    <ArtifactAuthorizationTransformationModel artifactRefId="CustomerPurchaseOrder" combiningAlgorithm="first_match">
        <ViewTransformationModel name="customer view">
            <ParticipantFilter expression="{user}.hasAppRole('Customer')" />
            <InformationTransformation>
                <SelectionWindow>
                    <Condition expression="CustomerPurchaseOrder/CustomerId={user}.id" />
                </SelectionWindow>
                <ProjectedAttribute xPath="/CustomerPurchaseOrder/CustomerId" />
                <ProjectedAttribute xPath="/CustomerPurchaseOrder/ManufId" />
                <ProjectedAttribute xPath="/CustomerPurchaseOrder/ProductCode" />
            </InformationTransformation>
            <LifecycleTransformation createPermitted="true">
                <CondensedState name="InProgress">
                    <OriginalState>WOCreated</OriginalState>
                    <OriginalState>MPOsSent</OriginalState>
                    <OriginalState>MPOsRecieved</OriginalState>
                </CondensedState>
                <TransitionAccessControl originalTransitionRef="CustomerPurchaseOrderLifecycle_Transition_6" description="Created-to-SentToManuf" />
                <TransitionAccessControl originalTransitionRef="CustomerPurchaseOrderLifecycle_Transition_15" description="Assembled-to-Canceled" />
                <TransitionAccessControl originalTransitionRef="CustomerPurchaseOrderLifecycle_Transition_14" description="MPOsRecieved-to-Canceled" />
                <TransitionAccessControl originalTransitionRef="CustomerPurchaseOrderLifecycle_Transition_13" description="MPOsSent-to-Canceled" />
                <TransitionAccessControl originalTransitionRef="CustomerPurchaseOrderLifecycle_Transition_12" description="WOCreated-to-Canceled" />
                <TransitionAccessControl originalTransitionRef="CustomerPurchaseOrderLifecycle_Transition_11" description="SentToManuf-to-Canceled" />
            </LifecycleTransformation>
        </ViewTransformationModel>
    </ArtifactAuthorizationTransformationModel>
</ApplicationAuthorizationModel>
```
Listing 1 - Authorization View XML example

2.3. I-Hub Main Components

This section describes the main components of the I-Hub prototype implementation. Figure 8 depicts the main logical components grouped into physical nodes, all exposed interfaces, and intended actor interactions with the system.

Figure 8 - I-Hub main components

The overall ACSI prototype system contains two physical node types: a Server and multiple Clients. All clients interact with the server using a web interface.

- **Server**: a machine that hosts artifact-based applications. The server component is deployed into a web server (e.g., Apache Tomcat). The server node contains the following logical components:

Figure 7 - Authorization View XML example
o **Siena**: a Java web application responsible for managing and running artifact-based applications. Siena’s engine is utilized in the ACSI framework as a single-silo engine that is agnostic to any segregation of execution responsibilities between multiple silos. Siena is extended with such capabilities using access control ‘extension points’ in the code. That is, at any point during the execution where multi-user execution authorization is a concern, the Siena engine consults with another logical model (i.e., the I-Hub), which is responsible for all authorization aspects. This is achieved by exposing the **Access-Control** interface in the illustrated architecture. For single-silo settings, **Siena** is shipped with default implementations that allow all participants to access all artifact instance features. This gives modularity to the implementation such that in single-silo settings, it is possible to easily detach the I-Hub as an external plug-in from the execution engine.

**Siena** exposes three REST-based interfaces to clients:
- **SienaEditor**: responsible for specifying all the components of an artifact-based application.
- **SienaManagement**: responsible for activating/deactivating artifact-based applications.
- **SienaRuntime**: exposes both artifact-type level (e.g., describeArtifact), and instance level operations, such as create, retrieve and invoke services.

As part of its underlying data-layer, the Siena engine uses the following data-sources:
- **Applications metadata**: the applications’ metadata (e.g., artifacts, services, information model, and lifecycle) are stored in XML files on the server.
- **Artifact-snapshots**: artifact instance information is stored in a relational database. The **dataaccess** interface abstracts the actual RDBMS using JDBC protocol. In our case, we use IBM-DB2.

- **I-Hub**: an extension to **Siena** for supporting participant authorization views. The I-Hub implements the **Access-Control** plug-in interface defined by **Siena**.
  - **IHUB Editor**: REST interface for specifying artifact authorization views per participant.

As part of its underlying data-layer, the I-Hub engine uses the following data-sources:
- **AuthorizationViews**: application views are stored in XML files that comply with the model illustrated in Figure 6. To validate the content of these files, the model itself was also implemented as an XML schema file (i.e., XSD) used as a reference for the validation.
- **LDAP Server**: stores all participant-related information. LDAP is a repository that is commonly used in industry for identity management. Administrators specify and maintain all user information directly in the
LDAP server. A UserDetailsService interface is used by the I-Hub engine to abstract the LDAP communication protocol.

- Client Browsers: end users interact with the I-Hub system using a browser. The client side is composed of three GWT modules (i.e., each module is a library of html, Java-scripts and images) as follows:
  - SienaEditor: UI module for editing and deploying artifact-based applications. This UI uses the server’s SienaEditor and SienaManagement interfaces.
  - IhubEditor: UI module for editing artifact authorization views. This UI uses the server’s IhubEditor interface.
  - DefaultUI: An optional UI module used for interacting with artifact application. Users (or other computerized systems) can also interact directly with the SienaRuntime server interface. The DefaultUI generates dynamic pages according to the artifact information-schema and lifecycle. The DefaultUI is useful for application debugging. However, in actual settings, we expect that client applications will interact directly with the SienaRuntime server interface. All DefaultUI interactions require user authentication. The DefaultUI consults the server for the artifact information-schema (i.e., Application Model Inquiry use-case). The server applies the relevant participant’s view constraints and returns the relevant portion of the artifact model.

2.4. I-Hub Internal Package Structure

Figure 9 illustrates the internal package dependency structure of the I-Hub component.
Figure 9 - Packages in the I-Hub component. Arrows represent a dependency relationship between packages.

The I-Hub component comprises the following packages:

- **model**: contains the authorization view realized as a class-model as defined in Figure 6. The classes are plain data objects annotated with JAXB (i.e., Java-Xml-Binding). These annotations specify how objects should be serialized/deserialized to/from XML documents. This is useful both for serializing objects to files (i.e., into the dataaccess layer) and for serializing objects to HTTP in the service layer.

- **dataaccess**: responsible for querying, retrieving, and saving authorization models. The current implementation uses XML files. One file per artifact-application.

- **editor**: business logic for editing authorization views. The editor interacts with Siena components to validate that the referenced artifact features actually exists. This ensures consistency between the application model maintained by Siena and the authorization model maintained by the I-Hub. In addition, the editor is responsible for validating the model and for generating corresponding error and warning messages.

- **service**: contains a REST interface for the IhubEditor. Incoming calls are forwarded to the editor package. This interface is implemented using JAX-RS specification (JSR 311) for the Java RESTful interface and uses the Jersey implementation.
• **users**: This package contains the participant’s object model and the `UserDetailsService` that is linked to the LDAP repository, as described in the previous sub-section.

• **runtime**: contains *Siena’s* access-control plug-in implementation. The plug-in is responsible for authenticating the current user and for granting corresponding access rights according to all applicable authorization-views.

## 2.5. The ACSI-Hub Framework – Runtime Demo

The following screen shots demonstrate two points in time during the execution of the Order-to-Cash application from two different perspectives, showing the implication of the authorization constraints being interpreted at runtime.

Figure 10 shows the perspective of a signed-in manufacturer user (Lior) and a signed-in supplier user (Rick), inquiring about their associated MPO artifact instances. It can be seen that while the manufacturer can manage two instances of this type, the same instances are not yet visible to the Supplier. This is the result of a ‘window’ constraint. Furthermore, each user can invoke different sets of service operations as the result of different ‘transition execution’ permissions. Similarly, a different set of input fields is available to each user, being the result of different ‘attribute-projection’ constraints. Finally, each user can access different artifact-types as a result of the omission of certain authorization views. The second screen in Figure 11 complements this example showing the same artifact instance being observed by two users, a customer and a manufacturer, each seeing the instance as being in a different state (i.e., ‘in-progress’ vs. ‘MPOs Received’). This is the result of a ‘condensation’ constraint being interpreted at runtime.
Current user is Lior. Lior is assigned with a Manufacturer role.

Current user is Rick. Rick is assigned with a Supplier role.

Figure 10 - Manufacturer/Supplier views

Current user is David. David is assigned with a Customer role.

Current user is Lior. Lior is assigned with a Manufacturer role.

Figure 11 - Customer/Manufacturer views

3.  **Appendix 1 – Example for Service Type Interface Description**

This section shows the concrete realization for a service descriptor using standard WSDLs.

**AddWorkOrderItems**

**ListofStrings AddWorkOrderItems** (StringMax75 token,

  Guid workOrderGuid,

  ListOfWorkOrderLineItems orderLineItems,

  [OptionSelection commonSelection])

**Throws** InternalFault,

AuthorizationFault,

*Requires ownership of the work order or UPDATE on WORK_ORDER_AND_LINE_ITEMS.*

Adds line items to a work order. Each work order item consists of information about a catalogue item that is intended to be ordered. You cannot use this operation to update an already existing work order item in the specified order. To update a work order item within a work order, use the operation UpdateWorkOrderItems. The operation will fail if any selected items are not permitted for purchase or have been deleted by the provider. In this case, these items cannot be added into the work order.

The following fields are ignored in the order items when they are added:
- LineItemID
- MPOID
- status

The OptionSelection input is optional when adding the item. The selection will be validated if present; however, it will be ignored if it is not present. When the work order is validated, any order items missing a required option selection will be rejected.

The item *Guids* returned by this operation will *not* be in any particular order; therefore a client should not assume that the *Guids* are in the same order as the items that were added. A client can use the *GetOrderItemNamesByOrder* operation to retrieve the pairing of catalogue item guids to order item guids.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token</td>
<td>Security token</td>
</tr>
<tr>
<td>workOrderGuid</td>
<td>GUID of the order to add items to</td>
</tr>
<tr>
<td>orderLineItems</td>
<td>Items to add to the order. The provider order GUID and order item state will be ignored. At least one order item is required.</td>
</tr>
<tr>
<td>commonSelection</td>
<td>A common option selection to apply to order items being added that do not have an option selection.</td>
</tr>
</tbody>
</table>

**Returns:**

*Guids* of the added order items
Request example

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:v10="http://echo.nasa.gov/echo/v10"
xmlns:typ="http://echo.nasa.gov/echo/v10/types">
  <soapenv:Header/>
  <soapenv:Body>
    <v10:AddOrderItems>
      <v10:token>securityTokenOfMax75characters0x77ff</v10:token>
      <v10:orderGuid>uniqueWorkOrderGUID</v10:orderGuid>
      <v10:orderItems>
        <!--Zero or more repetitions:-->
        <typ:Item>
          <typ:ItemGuid>catalogue_itemGUID_177</typ:ItemGuid>
          <typ:QuantityOrdered>700</typ:QuantityOrdered>
          <!--Optional:-->
          <typ:OrderItemDetail>
            <typ:OwningProviderOrderGuid>
              <typ:ProviderGuid>providerIBM</typ:ProviderGuid>
              <typ:OrderGuid>uniqueOrderGUID_2</typ:OrderGuid>
            </typ:OwningProviderOrderGuid>
            <typ:State>NOT_SET</typ:State>
          </typ:OrderItemDetail>
        </typ:Item>
      </v10:orderItems>
    </v10:AddOrderItems>
  </soapenv:Body>
</soapenv:Envelope>
```

Response example

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:v10="http://echo.nasa.gov/echo/v10"
xmlns:typ="http://echo.nasa.gov/echo/v10/types">
  <soapenv:Header/>
  <soapenv:Body>
    <v10:AddOrderItemsResponse>
      <!--Zero or more repetitions:-->
      <typ:Item>catalogue_itemGUID_177</typ:Item>
    </v10:AddOrderItemsResponse>
  </soapenv:Body>
</soapenv:Envelope>
```