Data-centric Web Services based on Business Artifacts

Roman Vaculín, Terry Heath, Richard Hull
IBM T.J. Watson Research Center, Hawthorne, NY 10532, USA
Email: {vaculin, theath, hull}@us.ibm.com

Abstract—Existing Web services standards consider data primarily on the level of inputs and outputs specifications, with the major focus on functional aspects of interactions. Majority of applications rely on data sources, but such data sources are not part of the Web service specifications and cannot be accessed directly by clients. The fact that data are treated independently or as second-class citizens severely limits re-use, flexibility, customization and integration options of current Web services. In this paper we suggest to extend the WS specifications by introducing a data-centric Web services model that integrates functional and data perspectives in one coherent framework.

The approach is based on Business Artifacts and in particular on the declarative modular Guard-Stage-Milestone (GSM) model. We introduce a Web Data- and Artifact-centric Service (W-DAS) model using GSM in its core which in addition to usual application specific WS operations defines a set of data access interfaces including CRUD operations, artifacts retrieval interface for querying, filtering and sorting data, and operations for arbitrary custom defined ad hoc run-time queries. We discuss W-DAS publish-subscribe mechanisms and implementation.

Keywords— data-centric web services, artifact-centric web services, business artifacts, services model

I. INTRODUCTION

Current Web Service (WS) technologies and standards largely focus on functional aspects of interactions with distributed components. With some level of simplification, a typical Web service can be seen as a collection of operations with each operation being a function/mapping like transformation that consumes several inputs and produces several outputs as a result. Specifications of such Web services primarily focus on definition of interfaces, the particular data structures exchanged (WSDL), the transportation serialization (SOAP), etc. While there exist variations (e.g., REST-full services) and extensions (e.g., Semantic Web services approaches such as OWL-S, SA-WSDL, SA-REST, WSMO, etc.) that consider additional aspects such as preconditions, effects, postconditions, semantic classification of the WS, etc., the functional substance remains more or less unchanged.

It turns out that very often the existing Web services (e.g., Amazon) rely on rich data sources, for example in the form of relational databases, which are only exposed to the clients in a rather limited way through predefined WS operations. Such operations typically provide access points to only certain very limited aspects of the underlying data sources. The problem of the current WS paradigm is that the specification of the underlying data-sources is not the focus of WS descriptors, which largely limits the possible clients benefits and WS integration and extension use cases. A parallel can be made between the current Web services and the current shape of the web pages vs. the deep web. On the web, majority of dynamic web pages and dynamic forms rely on “hidden” data sources [8] that are not well described as part of the web pages, and therefore data residing in such data sources are very hard to access in a systematic and automated way. To access the deep web data various techniques have been considered, for instance, to discover the data schema and the attributes domains [11], or to derive the likely data constraints by analyzing JavaScript client-side code [3]. While such approaches sort-of work, it is clear that these solutions are somewhat bizarre. To overcome some of the problems of deep web in the context of Web services and to fully benefit from providing Web services access points to clients, we argue in this paper that the definition of Web services should be extended to include a rich form of data sources specification. We propose to include a data schema specification (and possibly constraints) as an integral part of the WS, and we propose to also expose a variety of querying and data manipulation capabilities as part of WS interfaces.

In this paper we describe a data-centric approach to representation of Web services. Our approach is based on the established work from the area of Business Artifacts (BA, or “artifacts” for short) [13], [4] which provide foundations for systematic modeling and representation of process models integrated with data aspects on the schema as well as instance level. In the Business Artifacts paradigm, process models are defined as interactions of typically several key business artifacts with each artifact type consisting of its life-cycle (and life-cycle operations) and a data-schema (information model). As a lifecycle model we propose a Guard-Stage-Milestone (GSM) [9] model, a highly modular, declarative approach with a well-defined semantics [7] based on the structured ECA semantics. GSM supports hierarchical specification of process activities with embedded rich parallelism and as such it is suitable for tasks such as service composition, integration, and WS search/discovery. As a major contribution, we introduce a Web Data- and Artifact-centric Service (W-DAS) model using GSM in its core. In addition to usual domain specific WS operations, W-DAS model defines a set of data access interfaces including (1) CRUD (Create, Read, Update, Delete) operations for manipulating BA instances, (2) artifacts retrieval interface for querying, filtering and sorting data instances, and (3) operations for arbitrary...
custom defined ad hoc run-time queries. We discuss how the model can be applied using the publish-subscribe paradigm. We have implemented the proposed approach in the prototype system called Barcelona.

Organizationally, we start with discussing motivation, use cases and the running example. In Section 3 we introduce the Business Artifacts approach and the GSM model. Section 4 introduces the W-DAS process model and the data access interfaces. Section 5 discusses implementation. In Section 6 we review related work and in Section 7 we conclude.

II. Running Example and Motivation

Throughout the paper we illustrate our approach by examples referring to a dynamic, collaborative web-based questionnaire service, modeled based on a real application (the authors have developed several applications using the approach described in this paper). The questionnaire Web service provides operations for creation and management of dynamic structured questionnaires. A client of the service is provided an API for creation of her own questionnaires where each questionnaire consists of several questions of various types, possibly organized into several sections, with possibly one or more user roles responsible for answering the questions. The API supports creation and management of questions that can be reused across multiple questionnaires, and operations for running the questionnaire, including operations for checking the status of the questionnaire or particular questions, answering and re-answering the questions, and similar operations.

A. Use Cases and Benefits of W-DAS

The benefits of extending the WS specifications by including aspects of data schema are manifold. A fundamental benefit stems from the possibility of allowing the clients to interact with the Web service and its underlying data in much richer and more flexible ways than originally envisioned by the WS designer. Here we outline several use cases.

1) Querying ad hoc queries: Today, the WS clients can only take advantage of operations that a designer built at the design time. While such operations may rely on underlying data sources and may be implemented as some form of data query, clients cannot ask a variation of the query or, in general, to perform ad hoc queries against the data source, although the data exists and would allow it. It is beneficial to allow a Web service client to ask ad hoc queries against the underlying data sources employed by the Web service, as long as the query is permitted and supported by the underlying data source. (Example: In the questionnaire service, the client may want to define his own question retrieval and filtering operations, e.g., filter questions by people who are responsible for answering, editing, etc.)

2) On demand customization and function extensions: Being able to access the data model of WS data sources expands the customizations that the clients of a Web service can think of, and thus increases the re-use opportunities beyond the envisioned functions as defined by the WS designer. Exposing only the functional operations severely restricts customizations clients can make without modifying the original WS design (which is typically impossible). (Example: Consider a client of the questionnaire service wants to develop a new advanced monitoring and analytics capability over many questionnaires. This can be easily achieved by defining new ad hoc queries for summarizing and calculating statistics of questions statuses, etc.)

3) Loose data-driven integration: The traditional web services (WSDL/SOAP, REST) most often rely on the request-response or one-way message exchange pattern1. However, in many situations it is convenient to consider a looser type of integration employing content-based publish-subscribe paradigm. Exposition of the data schema as part of the public Web service specification will allow richer use of the publish-subscribe integration model with both the subscription conditions and content being driven by data schema and values. (Example: The advanced runtime monitoring capabilities may be defined by employing the subscriptions, with a condition referring to particular question category, user group or combination, and the notification updating the monitors status only when appropriate).

4) Application integration, migration provisioning in Cloud centric environments: In the context of Software as a Service (SaaS) where the entire software services are being moved to and hosted in cloud environments it is important that the WS specification addresses data aspects (e.g. the provided and required minimal data schema requirements and specifications) in addition to traditional functional interfaces, which allows easier integration, migration and provisioning.

III. Preliminaries: Business Artifacts

The business artifact centric approach [13], [4] considers data as an integral part of business processes models, and it defines the process model and its operations in terms of interacting key business artifacts (BAs). Each BA type is characterized by an information model and a lifecycle model. The information model records all relevant information about a BA instance as it moves through the business. The lifecycle specifies all possible evolutions of a BA instance over time. A BA instance of a particular BA type is an instantiated complex data type identified by a unique identifier, which can evolve over time as prescribed by its lifecycle. As an example BA, consider a bank CheckingAccount, which records relevant information about an account since it has been opened by a customer till it gets eventually closed and archived, with the lifecycle capturing all relevant states of the account, possible transitions, operations, etc. In this paper we employ the Guard-Stage-Milestones (GSM) model [9] for lifecycle representation of BAs. We give a brief overview here only with the level of details relevant for purposes of this paper.

1We note here that while WSDL specification supports notification message exchange pattern, it is rather underspecified and its use is limited.
Information model: We use the variant of the nested relation model as an information model. Information model of each BA type is a record type, where each attribute field is either a scalar, a record type, or a collection type. The records and collections can be nested arbitrarily.

GSM lifecycle: Compared to traditional lifecycle models, such as the finite state machine, the GSM model is substantially more declarative, supports parallelism in a single BA instance and hierarchical organization of lifecycle components.

The GSM lifecycle consists of stages. Stages provide a mechanism for structuring activities related to a BA instance. A stage is either composite or atomic. A composite stage may contain one or more nested substages. Atomic stages serve as placeholders for tasks and nesting is not permitted. Supported tasks include assignment (to/from attributes of the information model), one- or two-way service invocation, and BA instance creation (see [9] for details).

Each stage has one or more milestones and one or more guards. A milestone represents a named business-relevant operational objective, and in the information model it is represented as a Boolean attribute indicating if the milestone has been achieved or not. A milestone is associated with one or more expressions, referred to as sentries, of the form “on triggering event if condition” (both optional). A milestone gets achieved when one of its sentries becomes true (i.e., the triggering event gets detected and the condition is true). A milestone can also have associated a set of invalidating sentries which serve for invalidating the milestone, i.e., making it false. A guard is represented as an expression of the same form, i.e., a guard is a sentry. Conditions of sentries of milestones and guards range over the information model of the corresponding BA instance, and possibly of related BA instances. We employ an extension of the OMG Object Constraint Language (OCL) to represent the conditions.

Triggering events for milestones and guards might originate from the external world (e.g., from a human actor or an incoming service call) – referred to as external events, from other BA instance, or from internal processing of the lifecycle (e.g., as a result of milestones changing values, or stages changing their status) – referred to as internal or control events. Events are typed and may have a structured payload (data schema). Additionally, a stage or a milestone can be associated with a response handler which is a statement of the form “for e return EventType[event-constructor]” responsible for generating a response (return) event for a particular incoming event e.

The logic of stages operations is based on a flavor of an Event-Condition-Action semantics and it works as follows (for complete semantics see [7]). Each stage can be either active (opened) or inactive (closed). Stages get activated by means of their guards. A stage gets opened only if one of its guards gets triggered and only if its parent stage (if there is any) is active. When an atomic stage opens, its task is initiated automatically. The stage gets closed when one of its milestones is achieved or when its parent stage gets closed. Multiple stages of a single BA instance may run in parallel. When an incoming event causes a particular stage to open and such stage (or some of its milestones) has an attached response handler for this event, the response handler is activated when the stage closes (or the milestone is achieved). The response handler generates a response event correlated with the originating incoming event.

IV. DATA- & ARTIFACT-CENTRIC SERVICE MODEL

A data-centric service specification consist of (1) abstract interaction interfaces (APIs), which define the access points and operations available to clients, (2) service process model that integrates both process and data specification, and (3) access control specification which defines the access constraints of clients’ interactions. Figure 1 represents major components of the model. In the following paragraphs we discuss these components focusing primarily on the process model and service interfaces. The access control for data-centric process models has been addressed for instance in [10].

A. Data-Centric Process Model

Web service process models define how various service components work together, and how WS clients and other interaction participants interact with the service. Compared to traditional notion of process models embodied for instance in languages such as WS-BPEL, we expand the scope of the process model to embrace both the processing aspects as well as the data aspects and we refer to such process models as data-centric process models.

Definition 1. Data-centric process model D–PM is a tuple D–PM = ⟨DS, BS⟩ where DS is a data schema specifying the data aspects of the model, and BS is a behavioral schema specifying the processing aspects.

Definition of data-centric process model provides a generic minimum structure allowing different embodiments. For instance, a data schema may be a hierarchical data schema, or in the context of semantic Web services it may be represented as an RDF-S schema or as a set of RDF views [17]. We use business artifacts based on hierarchical data schema and GSM as an embodiment of a data-centric process model.

Before giving a formal definition, we first introduce notions of complex types (referred to as C), business artifact types (referred to as A), reference to artifact types (R), data schemas, and their corresponding instances. We use S to refer to a set of simple types (such as Int, String, Boolean, Float, etc.).

Definition 2. Complex/record type D = ⟨nameD, DS⟩, nameD is the name of the type, DS is a data schema of D.

2For simplicity we assume that event-constructor is a list of key-value pairs where key refers to an attribute defined in the data schema. We assume the same for artifact constructors and change sets used later in the text.
Definition 3. Business Artifact type $A = \langle \text{name}_A, \text{id}_A : \text{ARef}, \text{DS}_A, \text{LA}_A \rangle$, name$_A$ is the name of type $A$, id$_A \in \text{ARef}$ is an identifier of instances of $A$, with $\text{ARef}$ being a set of all possible identifiers of instances of $A$, $\text{DS}_A$ is the data schema of $A$, $\text{LA}_A$ is the lifecycle of $A$.

Definition 4. (Nested/Hierarchical) Data schema $\text{DS}$ is a set of named, typed attributes, i.e. $\text{DS} = \{\langle \text{name}_i : \text{type}_i, \text{cardinality}_i \rangle | i = 1, \ldots, k \}$, where $\text{name}_i$ is the name, $\text{type}_i \in T$ is the type, and $\text{cardinality}_i \in \{1..n\}$ is the cardinality of the $i$-th attribute. Set of types $T$ is a union of simple, complex, and references to entity types.

Definition 5. Artifact-centric process model $A$–$\text{PM}$ is a data-centric process model defined as a finite set of business artifact types, i.e. $A$–$\text{PM} = \{A_i | i = 1, \ldots, n \}$, $A_i \in \text{AT}$ is artifact type. GSM artifact-centric process model uses lifecycles expressed in GSM.

Depending on the particular implementation, the information model of each artifact type can be mapped directly to the schema of the datasource and materialized as the data instances, e.g. in relational database, or it may be defined as a virtual view over the schema of the data-source.

In the text we use the dot notation to refer to components of tuples, e.g., to refer to the name of $A$ we use $A$.name$_A$. Also, we use the notion of rooted path expressions to access components (attributes, etc.) of artifact and data instances.

Definition 6. Rooted path expression for an instance $x$ of a data type $D$ (or artifact type $A$) (or path expression rooted in $x$) is an expression $x.p_1.p_2 \ldots p_n$ defined inductively:
- if $x.p_1.p_2 \ldots p_{i-1}$ refers to (one or more) instance(s) of data type $D_{i-1}$ then $p_i$ can be one of the attributes (properties) defined in the data schema of $D_{i-1}$
- if $x.p_1.p_2 \ldots p_{i-1}$ refers to an instance of a tuple $(t_1, \ldots, t_k)$ (e.g., artifact instance) then $p_i$ can be one of tuple components $t_1, \ldots, t_k$

Example 1. Figure 2 shows the artifact-centric process model of the questionnaire service introduced in Section II. The process model consists of 3 artifact types: Questionnaire, Question, and QuestionAnswer. Questionnaire represents the data (e.g., unique identifier, questionnaire name, etc.) and the lifecycle supporting definition and running of questionnaires. Question provides representation of reusable questionnaire questions, with the lifecycle providing operations for editing of questions. QuestionAnswer represents an answer for a particular question associated with a particular questionnaire instance, with the lifecycle supporting primarily the answering activities. For each artifact type the figure shows information schemas and major stages with their guards and milestones.

In the context of the process model, the lifecycle of Questionnaire is shown as a sequence of stages, each stage represented as a rectangle with inputs, outputs, and relationships between stages. The stages are

- Stage 1: "Enter Question Info" with the operation "on Operation" meaning that the stage opens when an event of type QuestionInfo is received; it is an atomic stage with an assignment task $t$ in its body (which assigns the payload of $e$ to attributes of the artifact instance); and it closes when the milestone Info Entered is achieved (i.e., when the sentry "on t.onComplete()" is triggered). Info Entered milestone is associated with a response handler for event $e$ which generates an outgoing event of type Status correlated with $e$. The figure also captures some interactions between stages; yellow dashed arrows show sequencing inside the artifact, while black dotted arrows illustrate inter-artifact relationships.

B. Core Abstract Interaction Interfaces

Abstract interaction interfaces define end-points and operations that are available for clients. Our design goal is to provide a conservative extension of classical Web services while enabling rich, data-driven interactions and integration as outlined in the use case scenarios in Section II-A. As captured in Figure 1, there are four interfaces that a data- and artifact-centric makes available. We present an abstract specification of the interfaces in terms of operations, their inputs and outputs and corresponding types. We use an abstract notation $\text{operation-name(input}_1 : \text{Type}_1, \ldots, \text{input}_n : \text{Type}_n) : \text{OutputType}$ to specify operations. Depending on the particular implementation these interfaces can be implemented as concrete WSDL or REST APIs as discussed in Section V.

1) Execute/Transition Interface: This interface defines operations that directly interact with the business artifact lifecycles and lead to lifecycle transitions of one or more artifact instances. The execute/transition operations closely correspond to usual operations of classical Web services. The exact form and definition of operations and their inputs and
outputs, including all details such as WSDL types, messages, etc., are directly derived from the lifecycle specification of the process model. The mapping from the GSM artifact-centric process model is defined as follows.

**Derivation of Execute/Transition operations:**
Let \( A^{PM} = \{A_i | i = 1, \ldots, n \} \) be a GSM artifact-centric process model.
For each sentry of the form “**on e : EventType, if condition**” of some milestone or guard of some stage belonging to \( A_i \), \( L_{A_i} \)
derive a new unique WS operation as follows:
- **Operation name**: \( opName = \text{generate unique name based on } \text{EventType}_1 \text{ and } A_i \text{name}_{A_i} \)
- **Input parameters**: use event \( e \) and its payload as an input message; use the datatype \( \text{EventType}_1 \) as type of the input message
- **Output parameters**:
  - if for \( e \) there exists a response handler \( r \) in \( A_i, L_{A_i} \) of the form \( \text{for } e \text{ return } \text{EventType}_2[\text{event-constructor}], \text{make } opName \text{ a two-way operation (request-response)} \) with output message of the type \( \text{EventType}_2 \)
  - if there is no response handler for \( e \) make \( opName \) a one-way operation

Since we employ a nested hierarchical data schema for definition of all types, including the types of events, it is possible to represent such schema as an XSD data schema. This makes it particularly simple to define WS messages and their types by using the existing XSD definitions of data, event and artifact types. For RESTful services the operations can be defined in a similar manner.

**Example 2.** Consider the guard of Question artifact with sentry “**on e : QuestionInfo**” as illustrated in Figure 2. For this sentry a new operation is defined with name QuestionInfo, with the input message having the schema of QuestionInfo type, and with the output message having type of Status event type since there is a response handler defined for \( e \) that returns event of type Status. Invoking this operation leads to opening the “Enter Question Info” stage, and the response message is returned after the milestone “InfoEntered” is achieved.

2) **CRUD Operations:** CRUD operations serve for performing basic manipulations with artifact instances. We define a generic data object access interface that allows to create, read, update and delete artifact instances of a particular type.

**CRUD Operations:** Let \( A^{PM} = \{A_i | i = 1, \ldots, n \} \) be a GSM artifact-centric process model, \( R_T \) a set of references (identifiers) of artifacts

**Create** (type : \( A^{PM} \), artifact-constructor) : \( R_T \)
- creates a new instance of type type, with initial values specified by artifact-constructor, and returns an identifier (reference) of the instance

**Read** (type : \( A^{PM} \), id : \( R_T \), content-expression) : instance of type
- reads an artifact instance with type type and identifier id and returns the instance with the content defined by the content-expression (explained below)

**Update** (type : \( A^{PM} \), id : \( R_T \), change-set) : \( R_T \)
- updates the artifact instance of type type, with identifier id with values specified in the change-set

**Delete** (type : \( A^{PM} \), id : \( R_T \)) : \( R_T \)
- archives the artifact instance of type type, with identifier id, and returns a reference to an archived instance (some systems may support undelete operation)
3) **Artifacts Retrieval:** Predefined data retrieval API extends CRUD operations by enabling search, filtering and sorting of artifact instances and associated data in a relatively simple but flexible manner. Here we introduce the core generic operation for retrieval of artifact instances.

**Artifacts Retrieval:** Let \( A-PM = \{ A_i | i = 1, \ldots, n \} \) be a GSM artifact-centric process model

**RetrieveArtifactsList( type : A-PM, filter-expression, content-expression, sort-expression ) :** list of instances of type \( A_i \) retrieved by filter-expression that match the content-expression with the sort-expression defined by the content-expression.

- **filter-expression** is a Boolean expression testing whether an artifact instance should be returned in the results. We suggest it shall be a conjunction \( \land_{i=1} \cdots \land_{n} \phi_i \) where \( \phi_i \) has form \( ( \rho \text{ OP } \text{ value} ) \) with \( \rho \) being a path expression rooted in instance of artifact type \( A_i \) and \( \text{OP} \) being a relational operator (e.g., >, =, <, like, etc.)
- **content-expression** is a list of path expressions rooted in instance of artifact type \( A_i \) defining which attributes should be returned as part of the instance: we allow special symbols * and ** in content expressions which refer to all attributes of a given instance accessing either the direct values of the attributes only (* symbol) or all values including the transitive closure of the data graph (** symbol)

For a given artifact type \( A_i \in A-PM \) the API allows retrieval of artifact instances of this type \( A_i \), and of all data elements which are *transitively accessible* from the artifact instance of \( A_i \) by using its properties or the *references* to other artifact instances. The content expression can be used by the client to select only relevant attributes of the artifact. This definition allows a straightforward efficient implementation in querying languages (e.g., SQL, SPARQL, XPath, etc.). Notice, that this API does not support arbitrary join operations.

**Example 3.** Consider the retrieve operation call 

**RetrieveArtifactsList( 'QuestionAnswer', f, c, s )**

where

- **filter expression** \( f = \text{questionnaire}='67' \)
- **content expression** \( c = \{ \text{lastAccessed, question.* , actualAnswer.* } \} \)
- **sort expression** \( s = \text{sortBy question.questionID} \)

This call retrieves all instances of type *QuestionAnswer* belonging to questionnaire 67, answered by person “Smith”, belonging to Cars, Trucks, or Vehicles category, and with the *questionID* starting with “Q2”. The content of the returned instances will only consist of the value of *lastAccessed* attribute, all direct values of the *question* attribute, and all data transitively accessible using the *actualAnswer* attribute.

4) **Custom Data Access Operations:** Custom data access API addresses use-cases where simple artifact retrieval does not satisfy the needs of a client. For instances, such cases include situations where a more complex join needs to be performed against the data schema of the artifact-centric service. This API provides clients with a mechanism for defining new custom operations at runtime based on arbitrary queries that can be supported by the data schema. Interaction with this API has two phases. In the first phase, the client provides the information needed for creation of the new custom data access operation, such as the parametrized query, input parameters specification, and the output message type specification. Using this information, the artifact-centric service dynamically generates a new Web service operation using the provided query as an implementation access mechanism, with the corresponding input and output interfaces, and it returns to the client a unique reference for accessing the service. In the second phase, the client invokes the new operation providing the appropriate values of input parameters.

We distinguish two categories of custom data access operations, namely the *artifact query operations* which always return only instances of a given artifact type (similar to **RetrieveArtifactsList** but allowing complex queries with joins), and the *generic data query operations* which can return instances of any data type.

**Custom Data Access Operations:** Let \( A-PM = \{ A_i | i = 1, \ldots, n \} \) be a GSM artifact-centric process model

**CreateArtifactQueryOperation( opName : String, type : A-PM, query, input-params ) : ServiceRef**

- creates a new custom artifact query operation \( opName \) for retrieving instances of type \( A_i \) using query parameterized with input-params
- **input-params** is a set \( \{ (\text{name}_i : \text{type}_i | i = 1, \ldots, k) \} \), where \( \text{name}_i \) is a name of the *i*-th parameter, \( \text{type}_i \in \mathbb{S} \cup \mathbb{R} \) is type of \( i \)-th parameter (simple type or artifact reference)
- **query** is a parameterized query expression (e.g., in SQL) expressed with respect to the data schema of \( A-PM \), using the *input-params* as placeholders for actual values used during operation invocation (see Example 4)
- **ServiceRef** is structure containing unique identifier of the newly generated operations and an access-point specification (e.g., URL, etc.)

**CreateGenericQueryOperation( opName : String, query, input-params, output-type : C_T ) : ServiceRef**

- creates a new generic data query operation for retrieving instances of *output-type* using query parameterized with input-params
- **output-type** specifies type of the output message
- **DestroyQueryOperation( id : ServiceRef )** destroys the custom query operation

**Example 4.** The following operation call can be used by the client to define a new operation *GetDashboardData* for calculation of monitoring data, such as the overall number of questions in a given category answered by a particular user: 

**GetDashboardData(‘GetDashboardData’, q, inputs, ‘DashboardData’)**
After invoking this call, the artifact-centric service creates a new operation called GetDashboardData with two input attributes (ownerPar and categoryPar) and one output attribute of the DashboardData type (which contains at least the attribute CountAnsweredQAs).

While the query in Example 4 is rather simple, it clearly illustrates the generic principle that allows the clients to use arbitrary queries against the data schema. This substantially extends the integration and reuse options clients have.

C. Publish-Subscribe Interactions

Data- and artifact-centric process models substantially expand the ways in which clients can interact with Web services using a data-based (sometimes referred to as content-based) publish-subscribe paradigm. In the pub-sub approach the client subscribes to a provider and gets notified whenever a subscription is satisfied. Since the information model of W-DAS services is published, clients can conveniently express their subscriptions with respect to the information model.

<table>
<thead>
<tr>
<th>W-DAS publish-subscribe API:</th>
<th>Let $A$–$PM$ be a GSM artifact-centric process model</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateSubscription(</td>
<td>subscription-condition : sentry,</td>
</tr>
<tr>
<td></td>
<td>artifacts-selector, query-operation ) : Sub-ID</td>
</tr>
<tr>
<td></td>
<td>• creates a new subscription that gets activated when</td>
</tr>
<tr>
<td></td>
<td>subscription-condition is satisfied for artifact instances</td>
</tr>
<tr>
<td></td>
<td>defined by artifacts-selector; when activated it returns to the</td>
</tr>
<tr>
<td></td>
<td>client the result of query-operation</td>
</tr>
<tr>
<td></td>
<td>• subscription-condition is defined as a sentry (see Sec. III)</td>
</tr>
<tr>
<td></td>
<td>• artifacts-selector is used to identify artifact instances to</td>
</tr>
<tr>
<td></td>
<td>which the subscription applies; it can either be a specific</td>
</tr>
<tr>
<td></td>
<td>artifact instance identified by its identifier; or it can be a set</td>
</tr>
<tr>
<td></td>
<td>of artifact instances of a specific type $\in A$–$PM$ matching a</td>
</tr>
<tr>
<td></td>
<td>given filter-expression</td>
</tr>
<tr>
<td></td>
<td>• query-operation is one of the CRUD, Artifacts Retrieval, or</td>
</tr>
<tr>
<td></td>
<td>Custom Data Access operations</td>
</tr>
<tr>
<td>Unsubscribe( id : Sub-ID)</td>
<td></td>
</tr>
</tbody>
</table>

![Figure 3. Screenshot of Barcelona auto-generated GUI for accessing the Question artifact instances using the Artifacts Retrieval API](image)

V. IMPLEMENTATION AND DISCUSSION

A prototype engine, called Barcelona, has been developed to support modeling and execution of GSM based artifact-centric process models. It implements Web Data- and Artifact-centric Service model (W-DAS) using REST and WSDL APIs. Barcelona is an evolution of the Siena tooling [6] that supports an artifact model with state-machine lifecycles. Barcelona environment has three essential components:

1. Execution Engine provides support for management of business artifacts that are stored in the relational database; management of BA lifecycles; interactions with external environment via REST and WSDL service APIs; access control, and execution monitoring.

2. Solution Designer Editor is a light-weight web application covering most aspects solution designers need, including design of custom data access operations (the queries are expressed using parameterized SQL queries as illustrated in Example 4); the artifact-centric process models are stored in an XML format, with all datatypes defined as XSD schemas. To represent conditions of sentries Barcelona supports JEXL language and an extended version of OCL.

3. Default Runtime GUI is a web based tool for execution and testing of artifact centric services. It supports step through service execution and inspection, and it provides an auto-generated GUI for accessing the WS APIs. Figure 3 illustrates a fragment of the GUI for the RetrieveArtifactsList operation for the Question artifact with the list of artifact instances.

While our approach expands the scope of Web Services specifications, we believe that it does not necessitate more burden for service designers and developers. The reason is that the lifecycle and the information model need to always be addressed in application or business process development. W-DAS specifications in essence expose some specific aspects of the applications, and therefore the appropriate tooling, such as Barcelona, can generate in a model-driven manner vast majority of the W-DAS specifications from the existing application models. The authors have developed several applications using Barcelona and the GSM approach.

VI. RELATED WORK

Our work is based on a data-centric business artifacts paradigm [13], [4] with GSM being an evolution of the earlier artifact meta-models [6]. Recently different data-centric approaches have been proposed including a FlexConnect metamodel [14] where processes are organized as interacting business objects, or Case Management paradigm [18], but the major focus is not on Web services aspects. AXML Artifact model [12] is an alternative artifact centric approach based on a declarative form of artifacts using Active XML [1] as a basis with embedded service calls.
In the context of classical Web services there is related work focused on Data-centric Web services defined as those whose behavior is determined by their interactions with a repository of stored data. For example, in [15] a data modeling and contracting framework for data-centric Web services is proposed with data specified as entities and related constrains. The authors suggest CRUD interface, and in later work [16] they focus on verifying correctness properties for composition of services. Coming from the databases and grid world, in [2] Web services are used to access databases in a common middleware allowing uniform access to data resources. Our work considers richer interactions with services (e.g., custom data operations), and it provides an integrated view of the lifecycle and information model.

Various Semantic Web services approaches have been studied for representation of data-providing services using semantic views (e.g., RDF-views) [17] or a layered model employing syntactic and semantic layers [5]. The focus of [17] is on discovery of data-providing services (DPS), while [5] focuses on generating and managing DPS specification from its declarative definition. None of these works provides an integrated perspective as we do in this paper.

VII. CONCLUSIONS

In this paper we developed a data-centric model for representation of generic Web service. The approach uses Business Artifact and the Guard-Stage-Milestone model as its core. The clear strength of the proposed model is close integration between data and behavioral / functional aspects of the WS representation. We believe that it is critically important to elevate data, and in particular the data schemas of Web services, and give it the same importance as to the processing aspects. The reason is that by exposing the data and by allowing WS clients to interact with data more directly, in addition to the usual WS operations, many more re-use, integration, and customization scenarios will be possible compared what is possible today. Our approach is grounded in the GSM model that is declarative, flexible and modular and has well defined semantics. The variety of application interfaces for manipulation with data is designed in such a way that allows simple operations to be used very easily, and allows also performing of arbitrary ad hoc complex queries if needed. As a side-effect of making data available to clients, also the integration modes will change. In particular, the data-based publish-subscribe paradigm will become more prominent.

Our work provides foundations for several important directions for future work. The framework we developed allows flexible and rich ways of service composition and discovery. We have developed preliminary results in both of these directions, which are not reported in this work, but it is clear that fundamental research work needs to be done to develop sound composition and discovery approaches.

ACKNOWLEDGMENT

This research was partially supported by NSF grant IIS-0812578.

REFERENCES