Work as a Service Meta-model and Protocol for Adjustable Visibility, Coordination, and Control

Roman Vaculín, Yi-Min Chee, Daniel V. Oppenheim, and Lav R. Varshney
IBM Thomas J. Watson Research Center, USA
email: {vaculin, ymchee, music, lrvarshn}@us.ibm.com

Abstract—The Work-as-a-Service (WaaS) paradigm models work engagements as compositions of interconnected service requests, where there is a separation between the coordination of work and the actual work enactment. Here we revisit the WaaS conceptual meta-model and extend it to enable work decomposition and adjustable management/control of how work is coordinated and how it is done. In particular, we propose a specific WaaS protocol for decomposition, delegation, and control of work engagements, using ideas from the area of Business Artifacts. The goal is to enable simple communication and coordination between requestors and providers of work; and to support clear management and coordination during both planning and enactment of work. Importantly we introduce a new notion of a coordination lifecycle, consisting of loosely coupled milestones, domain-specific information attributes, and sets of abstract observable activities to be performed. Algebraic operations on coordination lifecycles when encapsulated service requests are torn, merged, paused, and resumed are defined and valid operations are specified. The meta-model and protocol are independent from the specific coordination enactment model which may employ centralized coordination, fully distributed coordination, or other models under various optimization objectives.

Keywords—work, coordination, milestones, business artifacts

I. INTRODUCTION

Managing, coordinating, and executing work in large-scale projects is intrinsically complex, especially when involving cross enterprise collaboration [1]. Many such projects do not meet planned schedules, budgets, or goals. Lack of communication and collaboration across silos along with poor visibility into work are major causes. To address this issue, we developed the Work-as-a-Service (WaaS) paradigm [2], which models work engagements as compositions of interconnected service requests. In the WaaS approach, the work service request encapsulates units of work that can be treated relatively independently. For example, a work service request can be assigned to a service provider, several work service requests can be combined together (“composed”) to fulfill a particular work engagement, or optimization techniques can be applied to identify the optimal composition, decomposition, and assignment of work. The promise of the WaaS approach is to bring more granular and detailed visibility into work progress. Various combinations of these three pieces can be used to faithfully model contracts between requesters and providers, providing the appropriate level of visibility, coordination, and control.

The WaaS protocol herein builds on the notion of the service requests specified by means of coordination lifecycles and enables hierarchical composition and decomposition of work service requests, enables safe delegation of work fragments, while maintaining the desired level of visibility and privacy. The proposed meta-model and protocol are independent from the specific coordination enactment model which may employ centralized coordination, fully distributed coordination, or other models.

The remainder of the paper is organized as follows. In Section II we review prior work on WaaS and on business artifacts. Next, in Section III, we discuss motivation scenarios.
Section IV gives an overview of our technical approach. Sections V and VI introduce the WaaS meta-model and its formalization, while Section VII defines algebraic operations for manipulation with WaaS packages. In Section VIII we discuss the coordination protocol and its variations, and in Section IX we conclude.

II. PRELIMINARIES

A. Work as a Service

This section summarizes the key concepts in [2]. The foundational idea behind work as a service is to take a services-oriented view of both the doing and the coordinating of work. The doing of work is encapsulated as a service request and the coordination of work involves routing service requests to work systems. Indeed, the basic construct in the WaaS framework is an encapsulated service request that isolates each service provider and facilitates the modular organization of work [5].

As depicted in Figure 1, a service request between a requestor (R) and a provider (P) contains two distinct parts: coordination information for coordinating work and payload information for doing work. A requestor is a service system that requests work to be done, provides inputs, and specifies the requirements. A provider is a service system charged with fulfilling the work request to meet requirements.

Coordination information deals with business concerns such as risk, cost, schedule, and value co-creation [6]. On the other hand, payload information defines the deliverables and provides what is needed to do the work, such as designs or use-cases. This general two-part decomposition leads to a paradigm of work as a two-way information flow between service systems, rather than as a business process that needs to be implemented or integrated between two organizations.

The encapsulated service request is meant to hide implementation details on how the provider does the work, e.g., tools, information technology infrastructure, data, and processes, from the requestor since the requestor need not care as long as requirements are met. The requestor may, however, specify specific visibility requirements of ongoing work that must be met. In particular, certain metrics or “sensors” that provide visibility on the doing of work may be instrumented and queried.

By thinking of encapsulated service requests as service blocks, one can decompose large work engagements into several smaller encapsulated service requests and conversely building up small encapsulated service requests into larger work engagements. Several basic operations arise with service blocks, including delegating blocks, tearing into several blocks, and merging blocks together. These basic operations can be used in combination to generate other structures, such as coordination hubs [7], [8].

Coming back to coordination, the goal is to satisfy the business concerns faced by both requestors and providers, as well as perhaps larger systematic concerns. Negotiation among these different perspectives is possible through the coordination information. Note that coordination may be carried out by the requestors and providers themselves or by an external coordinating agent, and that concerns from the different perspectives of requestors and providers may conflict.

The present paper will develop an explicit manifestation of coordination information using business artifacts and discuss how control, coordination, and visibility are enabled by this form of coordination information. Explicit methods for determining coordination information under operations like delegating, tearing, and merging, will also be discussed. Note that these basic operations are related to prior definitions in management theory [9, p. 140] and systems theory [10].

B. Business Artifacts

This section reviews basic concepts from the business artifact centric approach [3], [11], [12] to business processes. Note that other data-centric approaches to business process management have also been proposed [13]–[18].

The business artifact approach considers data as an integral part of business processes, and it defines the business processes 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 business-relevant information about a BA instance as it moves through the business. The lifecycle specifies all possible evolutions of a BA instance over time. As an example BA, consider a bank checking account that records all information about an account since it has been opened by a customer until it eventually gets 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 [19] for lifecycle representation of BAs. We give a brief overview here only with the level of detail relevant for the purposes of this paper.

1) Information Model: We use the variant of the nested relation model [20] 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.

2) GSM Lifecycle: 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. Various tasks are supported including assignment (to/from attributes of the information model), service invocation, human tasks (performed by human actors), BA instance creation, etc. (see [19] 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 of the form “on triggering event if condition” (both optional). A milestone gets achieved when one of its expressions becomes true (i.e., the triggering event gets detected and the condition is true).

A guard is represented as an expression of the same form. The conditions of expressions of milestones and guards range over the information model of the corresponding BA instance, and possibly of related BA instances. The conditions are expressed using an extension of the OMG Object Constraint Language (OCL) [21].

The triggering events for milestones and guards might originate from the external world (e.g., from a human actor or an incoming service call), from other BA instance, or from internal processing of the lifecycle, e.g., as a result of milestones changing values.

3) The Logic of Stage Operations: The logic of stage operations is based on a flavor of an Even-Condition-Action semantics and it works as follows (for complete semantics see [19]). 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 stages opens, its task is initiated automatically. When a composite stage opens, its immediate sub-stages become eligible for opening. The stage gets closed when one of its milestones is achieved or when its parent stage gets closed (in both cases the purpose of the stage was achieved). Notice that multiple stages of a single BA instance may run in parallel.

4) Note: Compared to traditional lifecycle models, such as the finite state machine variants, the GSM model is substantially more declarative, supports parallelism within a single BA instance and hierarchical organization of lifecycle components. All these features turn out to be important for defining WaaS specifications.

III. MOTIVATION AND USE-CASES

In this section we describe in more detail the problems and motivating use-cases of work engagements in collaborative enterprise environments. This motivates the WaaS approach, and in particular the need for the coordination lifecycles, their components, the basic operations for manipulation of coordination lifecycle, and for the coordination protocol.

Problems that arise in a collaborative enterprise environment have been discussed in [11], [22]–[24], along with the resultant need for agility to quickly reconstitute work in order to address new opportunities or disruptions due to the rapid change of pace in today’s business environment. Because of these factors, planning and coordinating work cannot be a static activity. It can’t be conducted only at the outset of a collaboration, but rather requires the ability to adapt on the fly. These issues have started to be addressed in Event-driven BPM approaches [25], [26].

In the context of WaaS, adaptation can take the form of modifications to both the structural and temporal characteristics of the work requests that constitute the collaborative effort. We introduce the basic structural (tear, merge) and temporal (pause, resume) operations which are motivated by responses to issues and opportunities that arise during collaborative work, and describe the resulting coordination problem for this dynamic environment. We use as an example cross-organizational application development and maintenance, although it should be obvious that the same principles can equally apply to other scenarios of distributed work.

Consider a globally distributed team of individuals collaborating to perform application development and maintenance work. Development will require many different types of skills, from design and implementation to test and various levels of support. Within a single organization, different types of skills might be clustered geographically, for example, to build up a critical mass of skills for fostering development of certain capabilities. The organization may have multiple instances of such centers of competency for a particular skill, for redundancy to avoid disruptions from infrastructure outages, natural disasters, or political unrest. Regulations, whether in the form of restrictions on the nationality of project participants or constraints on the movement and storage of sensitive data, might also dictate the need for multiple instances. This structure can also exist across a number of different organizations and enterprises collaborating together, where each strategically decides to focus on one or more specific skill areas, to the exclusion of others [27]–[29].

Suppose that a project is initially parceled into WaaS requests, and assigned to different organizational units for completion. Ideally, work will flow between the units as described by the initial project plan, and coordination consists solely of ensuring that deliverables (i.e., required inputs) are made available, and status is updated accordingly. However, experienced project managers understand that the ideal case is rarely encountered in practice [22].

One common issue that can arise during collaborative development work is that one unit falls behind in the work that was assigned to it. There may be several underlying reasons, e.g. requirements are unclear, a key team member leaves the organization, or becomes unavailable for various reasons, outage of supporting systems and infrastructure, etc. Regardless of the specific details, in the WaaS abstraction, this situation needs to become visible through coordination information that is exchanged through the WaaS packet. The coordination response can take one of many flavors.

One possible response is simply to accept the delay and
shift the scheduled end date of the package further into the future. This obviously impacts the start of other WaaS requests which depend upon the delayed deliverables. Additionally, if the work must be combined with perishable outputs that are being produced by another unit,\(^1\) it may be necessary to pause other WaaS requests which are currently ongoing and resume them later to ensure that they complete at the same time as the new scheduled end date of the tardy request.

These temporal operations on WaaS requests (pause & resume) allow shifting of work in time in order to accommodate schedule impacts on dependent work.

Another possible response to late work is to re-scope the work that is to be done by the lagging unit. This can be accomplished by tearing the WaaS request into smaller pieces, and re-assigning some of the sub-units to different organizational units. In this way, work can be done in parallel, resulting in schedule compression.

On the flip side, suppose an organizational unit is operating extremely efficiently, such that it is completing its assigned work faster than scheduled. This presents an opportunity for the collaborative work to be delivered sooner than planned. In order to do this, some work may need to be shifted earlier (which can be modeled as “negative” pause/resume).

In addition, it may be desirable to give the well-performing unit additional work to do. This can result in the merging of WaaS requests that were originally assigned to this unit with requests that were assigned to other units. Such merging can be particularly beneficial if it eliminates dependencies that existed between the original requests, or if it eliminates activities or milestones which are no longer needed after the merge (e.g. audit, quality control, context determination).

The issues and opportunities described above view coordination response from the perspective of managing work schedule. Issues and opportunities also arise from additional concerns such as quality or cost. For example, if a unit that is performing development work consistently delivers code with many defects, it may be desirable to re-scope or reassign work by tearing and merging, as described above, or it may simply be necessary to inject additional coordination information into the WaaS requests assigned to the unit.

Given this set of structural and temporal operations, we can then define the problem of coordination as one of determining when it is best to tear/merge/pause/resume work in order to achieve certain objectives, where the objectives may be to minimize cost or time, or maximize quality, etc.

IV. OVERALL APPROACH / ARCHITECTURE

The fundamental WaaS encapsulation separates the concerns of doing work from coordinating work [2]. The following sections will provide a rigorous definition of the WaaS meta-model and formalize various operations that are required to facilitate coordination. In this section we provide an intuitive description of how coordination can be managed through a conceptual entity we call a hub [7], [23]. The main purpose of a hub is to identify all issues that affect the ongoing operations and stated business objectives, to determine an optimal response, and to enact this response. Visibility facilitates the identification of issues and also provides the information and its context so that informed decisions can drive coordination. In theory coordination could be fully automated but in practice important decisions are made by people, especially in large projects where different potentially conflicting stakeholder concerns must be carefully balanced.

From a provider perspective the problem of coordination may be stated as how to maximize the fulfillment of all the service requests it receives through an optimal utilization of all available resources. At any given time a provider may receive many requests from multiple requestors. A provider may vary in granularity from an individual, team, or project, to an entire organization. A provider organization may itself comprise several organizations, such as business units within a single enterprise or a collaboration that brings together several enterprises to achieve a common business objective. But the fundamental problem of coordination remains unchanged regardless of the provider’s organizational granularity, albeit it may become increasingly simpler as the granularity decreases from an organization to an individual.

Figure 2 depicts a hub as might be utilized by a large service provider organization. The hub has a collection of WaaS templates, a pool of providers, and an operations team. Each WaaS template represents a certain type of work. Any payload or coordination information that does not change across invocations can be preconfigured in the template. The specific skills of each provider will qualify him to service one or more WaaS templates. When the hub determines that work is required it will instantiate the appropriate WaaS template, update its payload and coordination data according to the execution context, and then assign it to any qualified provider. The operations team also continuously coordinates all ongoing work.

Figure 3 depicts the lifecycle of requests for service made by customers of the hub. Each such request may result in a large project comprising many different types of work. The operations team would first parcel this request into the different WaaS components that are required to accomplish

---

\(^1\)An example of a perishable output in global application development is a machine learning model for business analytics that must describe a quickly changing competitive environment.
the overall work. This parceling will identify temporal and other constraints such as cost, quality, or geography. It will also specify dependencies between WaaS components, such as start-finish. Once a request has been parcelled, all of its WaaS components can be scheduled. A scheduler can optimize the allocation of each WaaS component to the most suitable available provider. Different policies may be utilized to prioritize different business objectives such as cost, quality, schedule, or geography. As resources become available the scheduler will assign them pending WaaS request for execution. The hub’s operation team will coordinate all ongoing work for all of its customers. Finally, as each unit of work is completed it is integrated into final deliverable that is returned to the customer upon completion.

This model of work and coordination can be used recursively to form a dynamic ecosystem of requestors and providers that span across organizational boundaries. In the example depicted in Figure 4 the large hub represents a car manufacturer that is in the process of developing a new car. The company has been manufacturing low end family cars and has decided to start a new line with a high performance electric sports sedan. The manufacturer charged its internal design division with the mission to design this new car. This is represented by the WaaS request from the manufacturer’s hub to its internal design division. The WaaS payload data specifies all the business requirements and inputs required for creating the design, and its coordination data specifies milestones and metrics that provide deep visibility into the design process. However, the design division soon discovered that it doesn’t yet have in house designers with the required skills. It therefore decided to outsource the design of two major components to external design companies, one that specializes in the design of high performance engines and another that specializes in the design of high end luxury cars. The division’s own hub then split its incoming WaaS request into two new requests that were assigned to external tier 1 providers. A similar decision was made by the external engine design provider to outsource the design of the carburetor to a highly specialized 2nd tier service provider. However, to the car manufacturer’s own hub all the outsourcing of work from its design division to external providers can remain opaque. This is possible because the WaaS protocols guarantees that everything specified by the manufacturer’s original request will be honored regardless of how the work is executed or the number of provider tiers.

V. WaaS REQUESTS: META-MODEL

Recall that in the WaaS paradigm a work engagement is modeled as a decomposition into encapsulated service requests which specify the work requirements and coordination details between the work requestors and providers. Here we develop details of the service requests meta-model, define the notion of coordination lifecycle, and provide appropriate formalization.

The service request consists of coordination information and of the payload information. Figure 5 symbolically depicts the WaaS meta-model, expanding upon Figure 1. Payload is modeled as a collection of information artifacts represented as a data schema, capturing information such as inputs provided by the requestor, e.g., requirements specifications, and outputs such as expected deliverables specification, etc.

Coordination information represents all relevant information for coordination between the requestor and the provider. It may include risks, costs, work specification, price, etc., and importantly we introduce a new notion of a coordination lifecycle as a key mechanism for specifying different coordination approaches between the requestor and the provider.

The coordination lifecycle consists of loosely coupled
- milestones,
- domain-specific information attributes, and
- abstract observable activities to be performed.

Milestones serve as observable work progress indicators and are named Boolean attributes with anticipated deadline times. Milestones may be decomposed hierarchically into submilestones. Milestones serve as a primary high level coordination mechanism and in many cases provide enough information needed to assess the progress of a particular WaaS instance.

When more detail is needed for coordination between the requestor and the provider, a set of domain-specific coordination information attributes may be defined to provide mechanisms for information exchange between service requestor and provider. Typically, the information attributes are metrics.
or progress/status reports. The method by which information is exchanged between the requestor and the provider may be defined by a reporting policy, which we discuss in Section VIII. Importantly, delivery of information attributes can sometimes be tied to milestone achievement. Vice versa, achievement of a milestone can depend on values of one or more information attributes.

Finally, the coordination lifecycle may specify abstract activities into which the WaaS request can be decomposed with the goal of providing more granular and detailed visibility into work progress. We represent such abstract activities as stages as introduced in Section II-B. The stages need not exactly represent how the provider will perform the work. Typically, the stages will only provide a high-level, loosely-defined specification of the basic WaaS breakdown structure according to which the provider should report the progress to the requestor. In addition to the visibility and improved tracking ability, the stage breakdown structure also allows the requestor (or an independent controller) to better address situations such as delays of the work, as will be discussed in the following paragraphs.

The defining characteristics of the proposed coordination lifecycle is its high flexibility which allows the meta-model to be used in wide variety of scenarios. Various combinations of milestones, coordination information attributes, and the stage-based breakdown structure can be employed to faithfully model contracts between requestors and providers, providing the appropriate level of visibility, coordination and control as needed.

For example, in the situation when the requestor is collabor-
As explained above, the definition of a WaaS service request is supposed to define a minimal content structure appropriate for specification of real work requests, and at the same time provide the flexibility to allow the use of the meta-model in a wide variety of scenarios. Consequently, only some elements of the meta-model will be used in a particular engagement, while others will remain empty.

Another goal of the meta-model is to allow the use of capacity planning and optimization techniques for both the initial allocation of resources and for systematically dealing with situations such as schedule delay or capacity surplus, see Section III. Next we formally define several generic operations for manipulating WaaS requests.

VII. WaaS REQUESTS: OPERATIONS

It is important to define generic operations for manipulating WaaS requests, e.g. for optimization by a coordination hub. We distinguish structural operations that manipulate with the structure of the WaaS request and in particular with the coordination lifecycle, and temporal operations that impact only the temporal aspects of the WaaS schedule. Without loss of essential generality we will focus on the coordination lifecycle part of the WaaS request, and will describe the impact of the operations on the remaining WaaS elements where appropriate.

A. Structural Operations

Various motivations exist for manipulating the structure of WaaS requests. For example a particular fragment of a WaaS request might need to be assigned to a new provider, an existing request might need to be broken into several requests to allow parallelization and speedup of a schedule, or contrariwise it may be convenient to merge several request to achieve economies of scale, or to deal with capacity reductions. See Section III.

Before giving details of specific structural operations, let us first introduce the notion of dependencies which will play an important role for capturing structural relationships between elements of the coordination lifecycle. We represent dependencies as first-order logic expressions.

Definition 1 (Dependency). Dependencies are expressions in language \( D \) which is a subset of first-order logic. Expressions in \( D \) are conjunctions consisting of the following predicates: \( \text{substage}(x, y), \text{reads}(x, y), \text{writes}(x, y), \text{dependsOnEventFrom}(x, y), \text{dependsOnValue}(x, y) \).

The truth value of an expression \( \varphi \) in \( D \) is evaluated with respect to a particular coordination lifecycle \( cl \) (denoted as \( cl \models \varphi \) if \( \varphi \) is true in \( cl \)). The truth value of predicates in \( D \) relative to \( cl \) is defined as follows:

- \( cl \models \text{substage}(x, y) \) iff \( x \) is a substage of \( y \)
- \( cl \models \text{reads}(x, y) \) iff stage \( x \) reads attribute \( y \)
- \( cl \models \text{writes}(x, y) \) iff stage \( x \) writes attribute \( y \)
- \( cl \models \text{dependsOnEventFrom}(x, y) \) iff truth value of \( x \) (stage guard, milestone) depends on event from \( y \)
- \( cl \models \text{dependsOnValue}(x, y) \) iff truth value of \( x \) (stage guard, milestone) depends on value of \( y \)

The truth value of complex expressions is defined by induction from the basic predicates as usual.

With this definition of dependency, we introduce two elementary structural operations—\textit{tear} and \textit{merge}. Intuitively, the operations work as follows. The operation \textit{tear} takes as input a coordination lifecycle, breaks it into two coordination lifecycles that are partitions of the original one, and generates a dependence expression capturing key relationships between the resulting lifecycles. The operation \textit{merge} takes two coordination lifecycles and dependencies between them as inputs and it produces a set of merged coordination lifecycles that satisfy the dependencies.

Definition 2 (Tear). The tear operation is a function

\[
\text{tear} : C \rightarrow \mathcal{P}(C \times C \times D)
\]

where \( C \) denotes a set of coordination lifecycles, \( D \) denotes dependencies between lifecycles, and \( \mathcal{P} \) symbol represents a power set of a given set.

Notice that for one coordination lifecycle there can exist many different ways of decomposition. The tear operation can generate a set of such possible decompositions. It is useful to restrict the tear operation to only such decompositions which are meaningful.

Definition 3 (Valid decomposition). Let \( (cl_1, cl_2, \delta) \in \text{tear}(cl) \) for a particular coordination lifecycle \( cl \). We call \( (cl_1, cl_2, \delta) \) a valid decomposition of \( cl \) if the following conditions hold:

1) milestones partitioning:
\( cl_1.M \cap cl_2.M = \emptyset \) and \( cl.M = cl_1.M \cup cl_2.M \)

2) stages partitioning:
   a) \( cl_1.S \cap cl_2.S = \emptyset \) and \( cl.S = cl_1.S \cup cl_2.S \)
   b) \( cl_1.sub = cl_1.sub|_{cl_1.S} \) and \( cl_2.sub = cl_2.sub|_{cl_2.S} \), i.e., the structure of substages in \( cl_1 \) and \( cl_2 \) is the same as in \( cl \) restricted to the appropriate stages set \( cl_1.S \) and \( cl_2.S \) respectively

3) valid dependencies:
   for every two stages \( s_i \in cl_1.S \) and \( s_j \in cl_2.S \):
   \( s_j \in cl.sub(s_i) \Leftrightarrow (\delta \Rightarrow \text{substage}(s_j, s_i)) \), i.e., \( s_j \) is a direct substage of \( s_i \) in \( cl \) if and only if \( \text{substage}(s_j, s_i) \) is part of dependencies \( \delta \)

4) attributes and deadline inclusion:
   \( cl_1.A = cl.A, cl_2.A = cl.A, cl_1.deadline = cl.deadline, cl_2.deadline = cl.deadline \)

When a particular tear function returns only valid decompositions for \( cl \) we call it a valid tearing function.

The high level recursive procedure \textit{computeValidTear} defined in Algorithm 1 can be used for computing a valid tearing function. The procedure follows the definition 3. The user of this procedure can provide an initial partial assignment of some stages from the lifecycle \( cl \) to each of the partitions \( cl_1 \) and \( cl_2 \), and the procedure will generate all possible valid decompositions of a coordination lifecycle \( cl \) which satisfy definition 3 and these provided partial assignments.
Algorithm 1: computeValidTear($cl, S_1, S_2, cl_1, cl_2, tear_v$)

**Inputs:**
- Coordination lifecycle $cl$
- Partial stages assignments $S_1, S_2$ to $cl_1, S, cl_2, S$, s.t. $S_1 \subseteq cl.S, S_2 \subseteq cl.S, S_1 \cap S_2 = \emptyset$

**Outputs:**
- A valid tearing function $tear_v$ respecting the partial stages assignments $S_1, S_2$

**Auxiliary variables:**
- $cl_1$ and $cl_2$ representing current decompositions

**Procedure:**
1) if $cl.S \neq \emptyset$ then // iterate over stages by using recursion
   - $s = \text{remove some stage from } cl.S$
   - if $s \notin S_1$ then
     - computeValidTear($cl, S_1, S_2, cl_1 + s, tear_v$)
   - if $s \notin S_2$ then
     - computeValidTear($cl, S_1, S_2, cl_1 + s, tear_v$)
2) if $cl.S = \emptyset$ then // termination condition; all stages partitioned to $cl_1$ and $cl_2$
   - compute $cl_1.sub, cl_1.M, cl_2.sub, cl_2.M$: // illustrated on $cl_1.sub, cl_1.M$
     - foreach $s_x \in cl_1.S$
       - add all milestones of $s_x$ to $cl_1.M$
       - let $s_\text{parent}_x$ be parent stage of $s_x$ (according to $cl.sub$)
       - if $s_\text{parent}_x = \text{null}$ then continue;
       - if $s_\text{parent}_x \in cl_1.S$ then
         - add $s_x$ to $cl_1.sub(s_\text{parent}_x)$
         - else add substage($s_x, s_\text{parent}_x$) to $\delta$
   - $cl_1.A = cl.A, cl_2.A = cl.A$,
     - $cl_1.\text{deadline} = cl.\text{deadline}$,
     - $cl_2.\text{deadline} = cl.\text{deadline}$
   - add ($cl_1, cl_2, \delta$) to $\text{tear}_v(cl)$; return;

This way, the user can for example select some stages that can be parallelized (and thus enable speeding up of the WaaS), and let the procedure to compute the remaining parts of the decomposition. In essence, the procedure first generates all valid stage decompositions which satisfy the partial stages assignment (section 1 of the algorithm), and subsequently for each decomposition it generates the $sub$ function, dependencies $\delta$ and and the remaining parts of the lifecycles $cl_1$ and $cl_2$ (section 2 of the algorithm).

Next, we define the $\text{merge}$ operation. In essence, the idea of $\text{merge}$ is to provide an inverse operation for $\text{tear}$.

**Definition 4 (Merge).** The $\text{merge}$ operation is a function

$$\text{merge} : C \times C \times D \to \mathcal{P}(C)$$

where $C$ denotes a set of coordination lifecycles and $D$ denotes dependencies between lifecycles. For each $cl_1, cl_2, \delta$ and $cl \in \text{merge}(cl_1, cl_2, \delta)$ the following holds:
1) milestones merging:
   - $cl.M = cl_1.M \cup cl_2.M$ and
   - $cl.\text{deadline} = cl_1.\text{deadline} \cup cl_2.\text{deadline}$
2) stages merging:
   - $cl.S = cl_1.S \cup cl_2.S$
3) attributes merging:
   - $cl.A = cl_1.A \cup cl_2.A$,

4) substages structure:
   - $cl.sub|_{cl_1.S} = cl_1.sub$ and $cl.sub|_{cl_2.S} = cl_2.sub$
5) satisfying dependencies: $cl \models \delta$

Notice that $\text{merge}$ is defined rather loosely in the sense that the merged coordination lifecycle $cl$ only needs to satisfy (imply) $\delta$, and similarly the $cl.sub$ function needs to agree with $cl_1.sub$ and $cl_2.sub$ only on the domain restricted to $cl_1.S$ and $cl_2.S$ respectively. Therefore, possibly many merged coordination lifecycles may be returned that satisfy such requirements. Sometimes, however, we need a more constrained notion of merging, as captured in the following definition.

**Definition 5 (Strict merge).** Let $\text{merge}, cl, cl_1, cl_2, \delta$ be the same as in Definition 4 and let $cl \in \text{merge}(cl_1, cl_2, \delta)$. The $\text{merge}$ function is called strict if it satisfies all conditions in Definition 4 with condition 5 substituted by condition 5’ as follows:

5’) prescriptive dependencies:
   - for every two stages $s_i \in cl_1.S$ and $s_j \in cl_2.S$:
     - $s_j \in cl.sub(s_i) \Leftrightarrow (\delta \Rightarrow \text{substage}(s_j, s_i))$

The following lemma establishes the desired inverse property of $\text{tear}$ and $\text{merge}$ operations.

**Lemma 6.** Let $\text{tear}_v$ be a valid tearing function and $\text{merge}$ be a strict merging function.
1) For each valid coordination lifecycle $cl$ and for every ($cl_1, cl_2, \delta$) in $\text{tear}_v(cl)$ the following holds:
   a) $cl \in \text{merge}_v(cl_1, cl_2, \delta)$
   b) $|\text{merge}_v(cl_1, cl_2, \delta)| = 1$, i.e., merge returns only one valid coordination lifecycle
2) For every $cl_1, cl_2, \delta$ and for every $cl \in \text{merge}_v(cl_1, cl_2, \delta)$ there exists $(cl_1, cl_2, \delta) \in \text{tear}_v(cl)$

**Proof:** Straightforward from Definitions 3 and 5.

**Lemma 7 (Computing strict merge).** For given $cl, cl_1, cl_2$ and $\delta$ let $\text{merge}$ compute $cl$ exactly according to conditions 1–3 of Definition 4, and let $cl.sub = cl_1.sub \cup cl_2.sub \cup \text{substages_}\delta^2$ where $\text{substages_}\delta$ is a function which is for stage $s \in cl.S$ defined as $\text{substages_}\delta(s) = \{s' | \delta \Rightarrow \text{substage}(s', s)\}$. Then merge is strict.

**Proof:** Straightforward by simply verifying the conditions of Definition 5.

**B. Temporal operations**

We consider two temporal operations, namely pause and resume.

**Definition 8.** Let pause be a function

$$\text{pause} : C \to C$$

and resume be a function

$$\text{resume} : C \to C$$

We abuse the notation a bit by applying the union operator to define $cl.sub$ function. The full definition of $cl.sub(s)$ for stage $s \in cl.S$ is as follows:

$$cl.sub(s) = \{s' | s' \in cl_1.sub(s) \lor s' \in cl_2.sub(s) \lor \delta \Rightarrow \text{substage}(s', s)\}$$
At minimum, for a coordination lifecycle $cl$ the $pause$ function modifies the value of the attribute $cl.A.pausedAt$ to the current time, and the $resume$ function modifies the value of the attribute $cl.A.resumedAt$ to the current time. Both functions update the status of $cl$ to paused or resumed correspondingly.

Intentionally, the $pause$ and $merge$ definition leaves out details which will be specific for a particular implementation and situation. Typically, in terms of the operational semantics, when $pause$ is applied to an WaaS request in progress, the ongoing activities will required to be stopped or paused if possible, or near finished activities may be finished promptly. Similarly, when $resume$ is applied to a paused WaaS request, the activities that can be resumed are expected to be resumed, and the unfinished activities that cannot be resumed are to be restarted. When necessary, the milestone achievement status and the milestone schedule might need to be updated appropriately taking into considerations the time when the WaaS request was paused.

VIII. COORDINATION PROTOCOL

Having described the various operations and decomposition techniques for WaaS units, we now discuss how to leverage these operations and formalisms to define a coordination protocol which can be used to manage the execution of work. In line with the different ways in which collaborative work can be organized and controlled [9], we recognize and support a variety of styles of coordination which are motivated by differences in both the temporal and structural characteristics of the coordination requirements.

From a temporal perspective, one aspect of coordination that can vary across different work engagements is the frequency with which coordination decisions must be made. This in turn dictates the kind of policy which is required for reporting the values of the coordination information. At one end of the spectrum, in the case of highly dynamic and time-sensitive collaborative work where management of the work requires constant adjustment, it may be necessary to employ real-time continuous reporting, whereby the attributes of a coordination lifecycle are presented to the requestor as a stream of values. An example would be continuous reporting on the rate at which a provider is processing some set of inputs, which may then drive decision-making on production of related items. Taking a page from continuous-time control theory, we can consider the coordination attributes as observable state variables and decision-making as the control policy to ensure appropriate system behavior [10].

At the other end of the spectrum, some work requires only very infrequent decision-making, which allows for a policy in which reporting of attributes is needed only when changes occur. This situation might apply, for example, to collaborative research projects that take place on a larger time scale. In this case, the coordination protocol can be entirely event-driven, where notifications of changing coordination attributes trigger an evaluation process which determines whether to apply merge, tear, pause, or resume operations. One can think about event-driven control theory [30] to describe such systems.

For some types of work, it is more practical for decision-making to occur on a regular periodic basis. In this case, batch reporting of coordination attributes on some pre-specified time interval can be appropriate, as is the case with many typical software development projects or maintenance activities. In such work, periodic coordination might be sufficient at most times, but exceptional conditions might still require a more immediate response. This can be addressed by a hybrid reporting policy, where batch reporting is employed as a general rule, with specific critical coordination attributes monitored on an as-changed basis. This has connections to discrete-time control theory.

Beyond these temporal characteristics, the WaaS coordination protocol is also influenced by the structural organization of the work. One such structural organization – the centralized hub model – was discussed in Section IV. In this model, the coordination protocol consists of the hub requesting work from each of its providers, receiving updated status of milestones and coordination attribute values (according to one or more of the reporting policies described previously), and determining and executing the set of operations which are needed to allow the work to progress in the best possible way. As mentioned previously, this hub model can also be applied recursively, where the provider of a WaaS unit employs its own internal hub using the coordination protocol just described to manage the work of its internal labor pool and/or suppliers.

As an alternative to the hub model, service systems may interact as part of a decentralized aggregation of requestors and providers working towards a common goal. In this structural model, pairs of service systems which interact as both requestors and providers to one another exchange coordination attributes and adjust their mutual requests as appropriate using the operations outlined above. In this scenario, any given participant may have multiple such collaborations going on simultaneously.

The nature of relationships between providers and requestors can also impact the collaboration protocol. For instance, consider two organizations which engage in cooperative competition, collaborating on some projects while competing on others. In this case, when applying tearing operations, the requestor may take the further step of tearing the information model as well, in order to avoid sharing additional information with the provider. This step can be omitted in a more open collaborative situation, in which case the entire information model is propagated with all of the requests which result from the tearing.

IX. CONCLUDING REMARKS

In this paper we introduced a meta-model for work as a service requests, and a protocol for coordination and control of work engagements involving cross enterprise collaboration. Furthermore we developed algebraic structural and temporal operations for manipulation with WaaS requests which allow to effectively address scenarios where work needs to be optimized, restructured, or rescheduled. We illustrated how
the merge and tear operations can be defined and how such operations can be implemented.

While we defined how valid WaaS coordination lifecycle decompositions look like there are other important considerations when it comes to applying such operations. Typically, the reason for tearing is to restructure the work to achieve benefits like getting the work done faster. Our current work presented in the paper provides formal foundations enabling such benefits. In the future work we plan to develop more advanced automated techniques that will enable fully or partially automated systematic handling of WaaS restructuring. We believe it is feasible to analyze the dependencies of possible WaaS decompositions, and in particular to leverage the level of dependencies among the decomposed WaaS request. For example, to achieve desired speed up of WaaS request it may be possible to automatically consider only such decompositions that allow high degree of parallelization, and thus multiple providers can get the overall work done faster. Similarly, our basic model can be modified to specifically support the coordination protocol variations. We have described in more detail the centralized form of the coordination protocol used in the context of hubs. The decentralized version may lead to variations of the proposed operations where only fragments of the information models and deadlines will be shared. Also, in the future work we plan to address the methods for measuring the effectiveness of Work-as-a-service and the presented meta model, including metrics focused on quality, speed, reduced coordination costs and overall cost of production.

ACKNOWLEDGMENT
Helpful conversations with colleagues within IBM Research are appreciated.

REFERENCES