Business Artifacts Discovery and Modeling

Zakaria Maamar[1], Youakim Badr[2], and Nanjangud C. Narendra[3]

[1] Zayed University, Dubai, U.A.E.
[2] INSA-Lyon, Lyon, France
[3] IBM Research India, Bangalore, India

Abstract. Changes in business conditions have forced enterprises to continuously re-engineer their business processes. Traditional business process modeling approaches, being activity-centric, have proven to be inadequate for handling this re-engineering. Recent research has focused on developing data-centric business process modeling approaches based on (business) artifacts. However, formal approaches for deriving artifacts out of business requirements currently do not exist. This paper describes a method for artifact discovery and modeling. The method is illustrated with an example in the purchase order domain.

Keywords: Artifact, Data, Discovery, Process, Operation.

1 Introduction

Continuous changes in market opportunities and conditions have led enterprises to re-engineer their business processes. Typically, these business processes are modeled in an activity-centric manner. While this way of modeling is popular, it has several limitations, e.g., aligning business requirements to business processes is not simple and modifying these processes mid-stream is cumbersome. A data-centric approach through *(business) artifacts* [3] can address these limitations.

As in our earlier work [2], we adopt the definition of artifact from [3] as a "*concrete, identifiable, self-describing chunk of information that can be used by a business person to actually run a business*". That is to say that an artifact is a self-describing collection of closely related data that represent a business record, which describes details of goods and services provided or used by the business. One would consider order and menu as artifacts when modeling a restaurant. An artifact is subject to changes that are reflected on a state transition system called $\mathcal{A}$rtifact $\mathcal{L}$ife-$\mathcal{C}$ycle ($\mathcal{ALC}$). Transitions between successive states in an $\mathcal{ALC}$ are the result of executing specific tasks in a business process.

There is an abundant literature on artifacts [1,3,4]. However, there is still a lack of rigorous approaches that assist those in charge of discovering and modeling artifacts We propose a method that examines the discovery of artifacts from three perspectives. The data perspective capitalizes on the data in a system and the dependencies between these data. The operation perspective capitalizes on the operations in a system and the dependencies between these operations. Out of the data and operation perspectives, two separate lists of candidate business

P.P. Maglio et al. (Eds.): ICSOC 2010, LNCS 6470, pp. 542–550, 2010.

artifacts start to emerge. Finally, the connection perspective establishes links between these two lists so that a list of final artifacts upon which the future system will be built is identified.

Our running example is a simplified purchase order scenario. A customer places an order of products via *Customer-App*. Based on this order, *Customer-App* obtains details on the customer's purchase history from *CRM-App*. Then *Customer-App* forwards these details to *Billing-App*, which calculates the bill based on this history (e.g., discount eligibility), and then sends the bill to *CRM-App*. The latter prepares the detailed purchase order and sends it to *Inventory-App* for order fulfillment. For the in-stock products, *Inventory-App* sends a shipment request to *Shipper-App*, which will deliver the products to the customer. For the out-of-stock products, *Inventory-App* sends a supply message to the requisite *Supplier-App*, which provides *Shipper-App* with the products for subsequent shipments to the customer.

The rest of this paper is organized as follows. Section 2 introduces the method along with a short description of the proof-of-concept prototype. Concluding remarks and future work elements are included in Section 3.

2 Our Method

Fig. 1 shows our proposed method for business artifacts discovery and modeling. The cloudy shape represents the business case-study. The ovals correspond to the beginning and end states of this method. The dashed bold-line rectangles correspond to the three perspectives mentioned in Section 1. The regular plain-line rectangles correspond to the steps to carry out in each perspective. The arrowed plain-lines connect the steps of the same perspective. Finally, the arrowed dashed-lines connect the steps of separate perspectives.

In the method, the data and operation perspectives rely on the description of the case study along with a complete understanding of other elements such as types of users and nature of business. The data perspective identifies the core

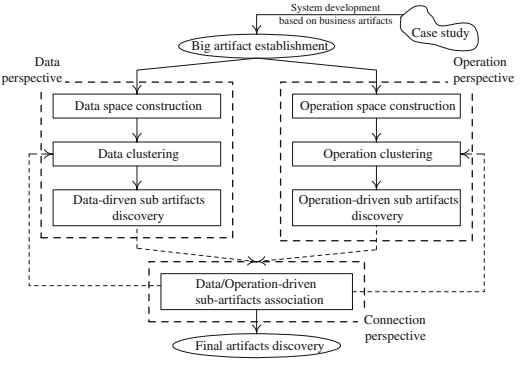


Fig. 1. Method for business artifacts discovery and modeling

data that are manipulated during users' needs satisfaction, and the dependencies between these data. This perspective leads into the *data space* of the case study. In parallel, the operation perspective identifies the core operations that are executed to satisfy users' needs, and the dependencies between these operations. This perspective leads into the *operation space* of the case study. Putting the data and operations together constitutes the basis for establishing different elements namely $\mathcal{B}$ig $\mathcal{A}$rtifact ($\mathcal{BA}$), $\mathcal{S}$ub $\mathcal{A}$rtifact ($\mathcal{SA}$), $\mathcal{F}$inal $\mathcal{A}$rtifact ($\mathcal{FA}$), $\mathcal{D}$ata $\mathcal{S}$pace ($\mathcal{DS}$), and $\mathcal{O}$peration $\mathcal{S}$pace ($\mathcal{OS}$).

The connection perspective connects the data-driven $\mathcal{SA}$s and the operation-driven $\mathcal{SA}$s together. By doing this, the $\mathcal{SA}$s are refined by (1) finalizing the structure of the $\mathcal{SA}$s in terms of data and operations, (2) identifying the possible interactions between the $\mathcal{SA}$s, or (3) either identifying new $\mathcal{SA}$s or combining some $\mathcal{SA}$s into other $\mathcal{SA}$s. Through this refinement, the $\mathcal{FA}$s are now available.

In the method, there is one $\mathcal{BA}$ that represents the business case-study to examine. A $\mathcal{BA}$ is like a melting pot that includes all the data and operations of the case study without any distinction to their types, natures, etc. $\mathcal{BA} = \langle \mathcal{DATA}, \mathcal{OPERATION} \rangle$ where $\mathcal{DATA}$ is a finite set of data $d_{i,i=1..n}$ and $\mathcal{OPERATION}$ is a finite set of operations $op_{j,j=1..m}$. □

2.1 Data Perspective

Establishing the data perspective requires three steps to be detailed hereafter.

`Data space construction space step.` The construction of a data space requires (1) extracting all the data $d_{i,i=0,\cdots,n}$ of the case study from $\mathcal{DATA}$ of the $\mathcal{BA}$, (2) pruning these data from synonyms, homonyms, and antinomes, and 3 describing each $d_i \in \mathcal{DATA}$ using $\langle id, l, t, i, u, e \rangle$ structure where: (id: is a unique identifier of the data), (l: is the $l$abel of the data), (t: is the $t$ype of the data whether atomic (e.g., real) or composite (e.g., address); in case of composite, identify recursively all the constituent data until the type of these constituent data is atomic; atomic is assigned by default to any data whose type is unknown), (i: is the $i$nput value of the data whether this value is assigned or calculated), (u: is the expected $u$se of the data, i.e., update, consultation, or both), and (e: is the r$e$strictions put on the receiver of the data, e.g., read only, no transfer, etc.). `Product`, `order`, `bill`, and `order_list_of_products` are examples of data extracted from the running example.

`Data clustering within the data space step.` To identify data-driven $\mathcal{SA}$s, the different data in the data space are grouped into clusters. A cluster is a potential candidate to be a data-driven $\mathcal{SA}$ although the mapping is not always one-to-one. For data clustering we use *dependencies* between data. We define the following dependency types: $\mathcal{U}$pdate $\mathcal{D}$ependency ($\mathcal{UD}$), $\mathcal{S}$ubstitution $\mathcal{D}$ependency ($\mathcal{SD}$), and $\mathcal{R}$emoval $\mathcal{D}$ependency ($\mathcal{RD}$). We refine each type of dependency into *strong* and *weak*. The *weak* type is motivated by the restrictions that can be put on data.

An $\mathcal{UD}$ from d_i to d_j exists if the successful update of d_i (i.e., value modification) triggers the update of d_j. $\mathcal{UD}$ is weak if the successful update of d_j is guaranteed regardless of the artifact that will host it once d_j is separated from

the artifact of d_i. $\mathcal{UD}$ is strong if the successful update of d_j is not guaranteed. By moving d_j to an artifact different from the artifact of d_i, there is a risk of not updating d_j (due to some restrictions). Both d_i and d_j have to remain in the same artifact.

A $\mathcal{SD}$ from d_i to d_j exists if the unavailability of d_i (due to some restrictions) makes d_j available for use. $\mathcal{SD}$ is weak if d_j is made available for consultation, only. d_j could be moved to a new artifact as long as there is no-strong $\mathcal{UD}$ from d_i to d_j. $\mathcal{SD}$ is strong if d_j is made available for both consultation and modification. Both d_i and d_j have to remain in the same artifact.

A $\mathcal{RD}$ from d_i to d_j exists if the deletion of d_i is declared complete subject to the successful deletion of d_j. $\mathcal{RD}$ is weak if the successful deletion of d_j is guaranteed regardless of which artifact hosts d_j and as long there is no-strong $\mathcal{UD}$ from d_i to d_j. $\mathcal{RD}$ is strong if the successful deletion of d_j is not guaranteed. By moving d_j to an artifact different from the artifact of d_i, there is a risk of not deleting d_j (due to some restrictions). Both d_i and d_j have to remain in the same artifact.

Before we continue describing the data clustering, some assumptions are made: (#1) If there is a dependency of type $\mathcal{X}$ from d_i to d_j, then a dependency of the same type from d_j to d_i will not be allowed, i.e., not commutative; (#2) If there exists a strong dependency of type $\mathcal{X}$ from d_i to d_j and another strong dependency of the same type from d_j to d_k, then a similar strong dependency from d_i to d_k will not be allowed or will be broken. The no-transitivity eases the ungrouping of data into separate clusters. Contrarily, if there exists a *weak* dependency of type $\mathcal{X}$ from d_i to d_j and another weak dependency of the same type from d_j to d_k, then a similar weak dependency from d_i to d_k will be formed. The transitivity eases the grouping of data into clusters; And, (#3) An artifact can limit the access to its data whether for consultation or update needs. This access concerns the operations that require artifacts in their processing.

We have developed two algorithms. The first algorithm identifies the necessary dependencies between the data of the data space. First, the algorithm determines whether the type of d_i is atomic or composite. If it is composite, then the data $\{d_{j,j\neq i}\}$ that make up d_i are determined. For each d_j, a $\mathcal{RD}$ from d_i to each d_j is then established. This is because the removal of d_i causes the removal of all its constituent data $\{d_j\}$. It is noted that if d_j had to be removed, this would make d_i incomplete calling for a $\mathcal{RD}$ from this d_j to d_i. However, this contradicts Assumption #1. If the type of d_i is atomic, then the algorithm determines whether the value of d_i is assigned or calculated automatically. If it is the latter, then the data $\{d_{j,j\neq i}\}$ that contribute to the calculation of the value of d_i are determined, and an $\mathcal{UD}$ from each d_j to d_i is established. In addition to the $\mathcal{UD}$, a $\mathcal{SD}$ is established from d_i to each d_j. Table 1 depicts some data dependencies from the running example. Strong (+) and weak (-) types of each dependency is done by the system engineer.

The second algorithm checks the consistency between the different dependencies. The purpose is to modify the respective types of these dependencies suitably from *strong* to *weak* and *vice-versa* so as to ensure that dependencies

Table 1. Some data dependencies before consistency checking

$d_i \rightarrow d_j$	pp	ota	**oda**	**omdd**	bn	ba	$\cdots$
order_date_approval (oda)	—	—	n/a	n/a	—	—	...
order_maximum_date_delivery (omdd)	—	—	**SD(-)**	n/a	—	—	...

Table 2. Some data dependencies after consistency checking

$d_i \rightarrow d_j$	$\cdots$	$\cdots$	oda	omdd	$\cdots$
omdd	—	—	**SD(+)**	n/a	...

are consistent and deadlock free. For example, in Table 2, the weak substitution dependency from omdd to oda is changed into strong (for the sake of clarity, Table 2 does not show all the dependencies.

`Data-driven` $\mathcal{SA}$`s discovery step.` Using Table 2, a *data dependency graph* is built. In this graph, each node is an atomic data, and each edge is a dependency with emphasis on update. Strong dependencies are depicted by plain lines, whereas weak dependencies are depicted by dashed lines. We, also, assume the existence of "default" strong dependencies between atomic data that belong to the same composite data, and this is done purely for consistency. These "default" dependencies are regarded as similar to strong update dependencies while determining the connected components of the graph; the connected components can be determined in linear time based on the number of nodes of the graph [5], and this is explained below. Fig. 2 shows an example of such a graph.

Fig. 2 shows that if the weak edges in the dependency graph are omitted, the resultant graph is a disconnected one, partitioned into one or more connected Clusters (C_i). Each cluster is a candidate to become a data-driven $\mathcal{SA}$. Connecting the different data-driven $\mathcal{SA}$s can happen through the dashed edges that symbolize weak dependencies. Later we show that the weak edges can be modeled via message passing between the different data-driven $\mathcal{SA}$s [2]. The interesting insight about Fig. 2 is that product and order composite data are merged into one cluster, i.e., one potentially separate data-driven $\mathcal{SA}$.

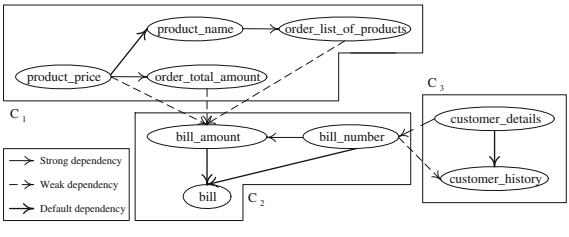


Fig. 2. Partial data dependency graph for the running example

Proposition 1. If weak edges are removed from a data dependency graph, then the graph is transformed into a disconnected one, with at least two connected clusters, (proof omitted).

Proposition 1 shows that each data dependency graph that contains at least one dashed edge can be partitioned into at least two connected clusters by removing the dashed edges. This proposition provides the justification for decomposing a data dependency graph into conceptually separate clusters, with each cluster depicting the data of a data-driven $\mathcal{SA}$, i.e., a "container" of the data that makes up the data-driven $\mathcal{SA}$. It should be noted that each data-driven $\mathcal{SA}$ is strongly coupled internally and weakly coupled externally, in line with well-known software engineering principles of high cohesion and low coupling. In this figure, $C_{1,2,3}$ can refer to *OrderContent*, *Bill*, and *Customer*, respectively.

2.2 Operation Perspective

Establishing the operation perspective of a case study requires three steps to be detailed hereafter.

`Operation space construction step.` The construction of an operation space requires (1) extracting all the operations $o_{i,i=0,\cdots,m}$ from $\mathcal{OPERATION}$ of the $\mathcal{BA}$, (2) pruning these operations' names from synonyms and homonyms, and (3) describing each operation using $\langle I, O, P, E\rangle$ structure, where: (I: is a set of input data reported in $\mathcal{DATA}$ of the $\mathcal{BA}$), (O: is a set of output data reported in $\mathcal{DATA}$ of the $\mathcal{BA}$), (P: is a set of preconditions, which are boolean formulae that must hold true if operation o_i is to start execution; these formulae are in CNF,and (E: is a set of effects, which are boolean formulae that hold true after operation o_i finishes execution; these formulae are in CNF as well).

We say that o_j is dependent on o_i if one of the outputs of o_i is an input for o_j, and if $e_i \in E$ is an effect of o_i and $p_j \in P$ is a precondition of o_j, then $e_i \Rightarrow p_j$; i.e., the effect e_i subsumes the pre-condition p_j.

We therefore define an *operation dependency graph* as a graph whose nodes correspond to operations, and whose edges correspond to dependencies between operations. Some operations from the running example are (o_1: place_order_submission), (o_2: customer_history_verification), and (o_3, bill_preparation).

`Operation clustering within the operation space step.` With an operation dependency graph in place, we have developed an algorithm to cluster operations. It groups the operations that use the same set of input data into clusters and then, relates these clusters to potential operation-driven $\mathcal{SA}$.

To illustrate the operation clustering algorithm, Fig. 3 depicts the operation dependency graph for some operations identified earlier. Initially, all the nodes belong to cluster C_1. Each operation o_i has a unique index starting with 1. The algorithm treats o_1 and checks its successor, i.e., o_2. Since both operations share common input data (namely order_list_of_products), o_2 is assigned to C_1. The algorithm continues with the successors of o_2, i.e., o_3, and so on until all the operations are processed. The various clusters created by the algorithm are depicted as C_1', C_2', and C_3' in Fig. 3. The algorithm takes $O(n)$ time, where n is the number of nodes in the operation dependency graph. In this figure, plain

lines represent intra-dependencies between operations in the same clusters, while the dashed lines represent inter-dependencies between operations in separate clusters.

`Operation-driven sub-artifacts discovery step.` Each cluster derived using the operation clustering algorithm is a potential operation-driven $\mathcal{SA}$. The reason is to link the operations that manipulate a common set of data into a common container.

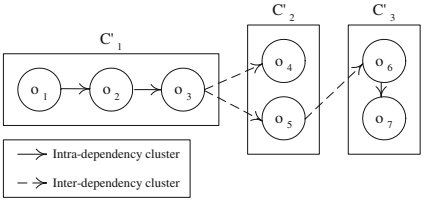
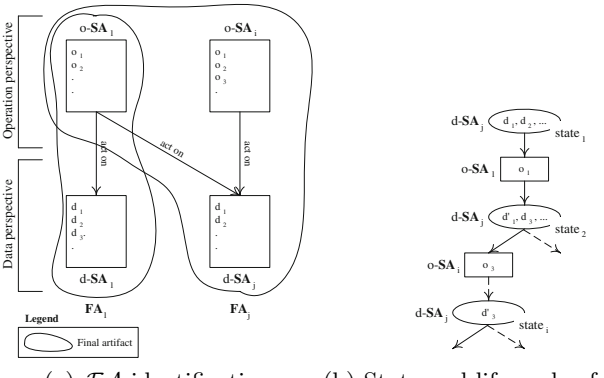


Fig. 3. Partial operation dependency graph for the running example

2.3 Connection Perspective

In this perspective the data-driven $\mathcal{SA}$s and operation-driven $\mathcal{SA}$s obtained previously are combined to form the future data-driven $\mathcal{FA}s$ of the future system. Data in the $\mathcal{DATA}$ set are common to both $\mathcal{SA}$s. The idea of the connection perspective is that the data-driven $\mathcal{SA}$s provide the necessary data, and the operation-driven $\mathcal{SA}$s provide the necessary operations that act on these data.

Fig. 4 shows the connection perspective that leads into obtaining $\mathcal{FA}$s. In Fig. 4 (a) each data-driven $\mathcal{SA}$ (e.g., d-$\mathcal{SA}_1$) along with its related operation-driven $\mathcal{SA}$ (e.g., o-$\mathcal{SA}_1$) form a specific $\mathcal{FA}$ for instance $\mathcal{FA}_1$. As a result if n data-driven $\mathcal{SA}$s exist in the data perspective, the same number of $\mathcal{FA}$s will exist too. To obtain the states and life cycle of a $\mathcal{FA}$ we capitalize on the input and



(a) $\mathcal{FA}$ identification (b) State and life cycle of $\mathcal{FA}_2$

Fig. 4. Connecting the data and operation perspectives

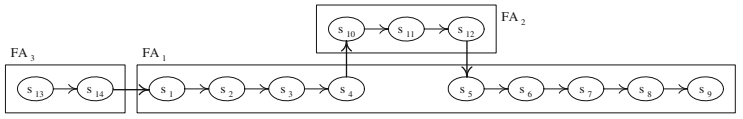


Fig. 5. Final artifacts

output data (e.g., $d\text{-}\mathcal{SA}_1.\{d_1, d_2, d_3\}$) that the operations (e.g., $o\text{-}\mathcal{SA}_1.\{o_1, o_2\}$) consume and produce, respectively. A state represents the data that are taken as input and modified later by an operation as output. As per Fig. 4 the intra-dependency between operations constitute the life cycle of a $\mathcal{FA}$, whereas the inter-dependency between operations constitute message passing between the respective life cycles of the different $\mathcal{FA}$s [2].

The set of final artifacts' lifecycles - modeled via their states - is depicted in Fig. 5. Operation o_1 is represented via $\mathcal{FA}_3$, and refers to the customer placing the purchase order. Further processing of the purchase order, described as per operations o_2 and o_3, are represented by states s_1 through s_4 in $\mathcal{FA}_1$. Operations o_4 and o_5, which deal with billing, are represented with sub-artifact $\mathcal{FA}_2$, after which control returns to $\mathcal{FA}_1$ for fulfilling the purchase order via states s_5 through s_9. These latter states pertain to operations o_6 and o_7.

2.4 Proof of Concept Implementation

We are now validating our business artifacts discovery and modeling method through a proof-of-concept prototype. It is implemented in Java on top of Eclipse 3.5.2. *Data Perspective*, *Operation Perspective*, and *Connection Perspective*

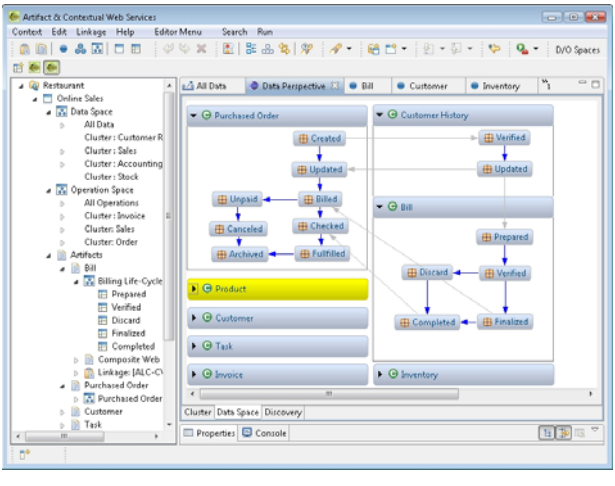


Fig. 6. Data clustering snapshot

modules constitute the prototype. Each module is accessible via interfaces based on SWT/JFace components, and a graphical editor built on GEF/EMF Frameworks. Currently we can model the data and operation perspectives, run the algorithms, and derive the final artifacts (Fig. 6).

3 Conclusion

Activity-centric modeling approaches, which mainly deal with business processes, fail to quickly respond to business changes. However, data-centric modeling approaches stand out as a serious alternative to model businesses by only focusing on data instead of activities. Data are often well-defined and stable regardless of the activities that manipulate them. Based on user requirements, our discovery and modeling method identifies business artifacts. The method applies a bottom-up analysis to assess and then gather fine-grained data into clusters before defining business artifacts. Concurrently, operations are identified and gathered into clusters with respect to their common input and output data. The final step consolidates data and operations. In term of future work, three areas are identified: compliance analysis of derived business artifacts with user requirements, impact of changes in user requirements on these artifacts, and exception handling analysis.

Acknowledgements. The authors thank Anil Nigam for his feedback.

References

1. Liu, R., Bhattacharya, K., Wu, F.Y.: Modeling Business Contexture and Behavior Using Business Artifacts. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 324–339. Springer, Heidelberg (2007)
2. Narendra, N.C., Badr, Y., Thiran, P., Maamar, Z.: Towards a Unified Approach for Business Process Modeling Using Context-Based Artifacts and Web Services. In: Proceedings of SCC 2009, Bangalore, India (2009)
3. Nigam, A., Caswell, N.S.: Business Artifacts: An Approach to Operational Specification. IBM Systems Journal 42(3) (2003)
4. Santhosh, K., Rong, L., Wu, W., Frederick, Y.: On the Duality of Information- Centric and Activity-Centric Models of Business Processes. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 32–47. Springer, Heidelberg (2008)
5. West, D.B.: Introduction to Graph Theory, 2nd edn. Prentice-Hall, Englewood Cliffs (2000)