Integrated Data and Process Management: Finally?

Marlon Dumas

University of Tartu, Estonia
marlon.dumas@ut.ee

Abstract. Contemporary information systems are generally built on the principle of segregation of data and processes. Data are modeled in terms of entities and relationships while processes are modeled as chains of events and activities. This situation engenders an impedance mismatch between the process layer, the business logic layer and the data layer. We discuss some of the issues that this impedance mismatch raises and analyze how and to what extent these issues are addressed by emerging artifact-centric process management paradigms.

1 The Data Versus Process Divide

Data management and process management are both well-trodden fields – but each in its own way. Well-established data analysis and design methods allow data analysts to identify and to capture domain entities and to refine these domain entities down to the level of database schemas in a seamless and largely standardized manner. Concomitantly, database systems and associated middleware enable the development of robust and scalable data-driven applications, while contemporary packaged enterprise systems support hundreds of business activities on top of shared databases.

In a similar vein, well-documented and proven process analysis and design methods allow process analysts to identify and to capture process models at different levels of abstraction, ranging from high-level process models suitable for qualitative analysis and organizational redesign down to the level of executable processes that can be deployed in Business Process Management Systems (BPMS).

But while data management and process management are each well supported by their own body of mature methods and tools, these methods and tools are at best loosely integrated. For example, when it comes to accessing data, BPMS typically rely on request-response interactions with database applications or packaged enterprise systems. Typically, data fetched from these systems are copied into the "working memory" of the BPMS. The data in this working memory are then used to evaluate business rules relevant to the execution of the process, and to orchestrate both manual and automated work. But the burden of synchronizing the working data maintained by the BPMS with the data maintained by the underlying systems is generally left with the developers.

More generally, the "data vs. process" divide leads to an impedance mismatch between the data layer, the business logic layers and the process layer, which in the long run, hinders on the coherence and maintainability of information systems. In particular, the data vs. process divide has the following effects:

- *Process-related and function-related data redundancy*. The BPMS maintains data about the *state* of the process, since these data are needed in order to enable the

system to schedule tasks, react to events and to evaluate predicates attached to decision points in the process. On the other hand, data entities manipulated by the process are stored in the database(s) underpinning the applications with which the BPMS interacts. Hence, the state of the entities is stored both by the BPMS and by the underlying applications. In other words, data are managed redundantly at the database layer and at the process layer, thereby adding development and maintenance complexity.

– *Business rules fragmentation and redundancy*. Some business rules are encoded at the level of the business process, others in the business logic layer (e.g. using a business rules engine) and others in the database (in the form of triggers or integrity constraints). Worst, some rules are encoded at different levels depending on the type of rule and the data involved. This fragmentation and redundancy hampers maintainability and potentially leads to inconsistencies.

The effects of this mismatch are perhaps less apparent when a one-to-one mapping exists between the instances of a given process and the entities of a given entity type. This is the case for example of a typical invoice handling process where one process instance (also called a "case") corresponds exactly to one invoice. In this context, the state of a process instance maps neatly to the state of an entity. Ergo, the data required by the process, for example when evaluating branching conditions, is restricted to the data contained in the associated entity (i.e. the invoice in this example) and possibly to the state of other entities within the *logical horizon* [5] of the said entity – e.g. the Purchase Order (PO) associated to the invoice. Accordingly, collecting the data required for evaluating business rules required by this process is relatively simple, while synchronizing the state of the process instance with the state of its associated entity (at the business logic and data layers) does not pose a major burden.

The impedance mismatch however becomes much more evident when this one-to-one correspondence between processes and entities does not hold. Consider for example a shipment process where a single shipment may contain products for multiple customers, ordered by means of multiple purchase orders (POs) and invoiced by means of multiple invoices – perhaps even multiple POs and multiple invoices per customer involved. Furthermore, consider the case where the products requested in a given PO are not necessarily sent all in a single shipment, but instead may be spread across multiple shipments. In this setting, the effects of a customer canceling a PO are not circumscribed to one single instance of the shipment process. Similarly, the effects of a delayed shipment are not restricted to single PO. Consequently, business rules related for example to cancellation penalties, compensation for delayed deliveries or prioritization of shipments become considerably more difficult to capture, to maintain and to reason about, as exemplified in numerous case studies [1, 9, 8, 3]. Traditional process management approaches quickly hit their limit when dealing with such processes. The outcome of this limitation is that a significant chunk of the "process logic" has to be pushed down to the business logic layer (e.g. in the form of business rules) – which essentially voids the benefits of adopting a structured process management approach supported by a BPMS.

Service-oriented architectures (SOAs) facilitate the inter-connection of applications and application components. Their emergence has greatly facilitated the integration of data-driven and process-driven applications. SOAs have also enabled packaged enter-

prise software vendors to "open the box" by providing standardized programmatic access to the vast functionality of their systems. But per se, SOAs do not address the problem of data and process integration, since data-centric services and process-centric services are still developed separately using different methods. A case in point is Thomas Erl's service-oriented design method [4], which advocates that process-centric services should be strictly layered on top of data-centric (a.k.a. entity-centric) services. Erl's approach consists of two distinct methods for designing process-centric services and entity-centric services. This same principle permeates in many other service-oriented design methods [7]. Such approaches do not address the issues listed above. Instead, they merely reproduce the data versus process divide by segregating data-centric services and process-centric services.

2 The Artifact-Centric Process Management Paradigm

This talk discusses emerging approaches that aim at addressing the shortcomings of the traditional data versus processes divide. In particular, the keynote discusses the emerging artifact-centric process management paradigm [1, 2] and how this paradigm, in conjunction with service-oriented architectures and associated platforms, enable higher levels of integration and higher responsiveness to process change.

Mainstream process modeling notations such as BPMN can be thought as being *activity-centric* in the sense that process models are structured in terms of flows of events and activities. Modularity is achieved by decomposing activities into subprocesses. Data manipulation is captured either by means of global variables defined within the scope of a process or subprocess, or by means of conceptually passive *data objects* that are created, read and/or updated by the events and activities in the process. In contrast, the database applications and/or enterprise systems on top of which these processes execute are usually structured in terms of objects that encapsulate data and/or behavior. This duality engenders the above-mentioned impedance mismatch between the process layer and the business logic and data layers.

In contrast, *artifact-centric* process modeling paradigms aim at conceptually integrating the process layer, the business logic and the data layer. Their key tenet is that business processes should be conceived in terms of collections of *artifacts* that encapsulate data and have an associated lifecycle. Transitions between these states in this lifecycle are triggered by *events* coming from human actors, modules of an enterprise system (possibly exposed as services) and possibly other artifacts, thus implying that artifacts are inter-linked. In this way, the state of the process and the state of the entities are naturally maintained "in sync" and business processes are conceived as network of inter-connected artifacts that may be connected according to N-to-M relations, thus allowing one to seamlessly capture rules spanning across what would traditionally be perceived to be multiple process instances.

The talk also discusses ongoing efforts within the Artifact-Centric Service Interoperation (ACSI) project[-2]. This project aims at combining the artifact-centric process management paradigm with SOAs in order to achieve higher levels of abstraction during business process integration across organizational boundaries. The key principle of

[-2] http://www.acsi-project.eu/

the ACSI project is that processes should be conceived as systems of artifacts that are bound to services. The binding between artifacts and services specifies where should the data of the artifact be pushed to, or where it should be pulled from, and when. In the ACSI approach, process developers do not reason in terms of tasks that are mapped to request-response interactions between a process and the underlying systems. Instead, they reason in terms of artifacts, their lifecycles, operations and associated data. Artifact lifecycles are captured based on a meta-model – namely Guard-Stage-Milestone (GSM) – that allows one to capture behavior, data querying and manipulation in a unified framework [6].

Upon this foundation, the ACSI project is building a proof-of-concept platform that supports the definition and execution of artifact-centric business processes. Challenges addressed by ACSI include the problem of reverse-engineering artifact systems from enterprise system logs – for the purpose of legacy systems migration – and the verification of artifact-centric processes, which by nature are infinite-state systems due to the tight integration of processes and data.

Acknowledgments. This paper is the result of collective discussions within the ACSI project team. Thanks especially to Rick Hull for numerous discussions on this topic. The ACSI project is funded by the European Commission's FP7 ICT Program.

References

1. Kamal Bhattacharya, Nathan S. Caswell, Santhosh Kumaran, Anil Nigam, and Frederick Y. Wu. Artifact-centered operational modeling: Lessons from customer engagements. *IBM Systems Journal*, 46(4):703–721, 2007.
2. David Cohn and Richard Hull. Business artifacts: A data-centric approach to modeling business operations and processes. *IEEE Data Eng. Bull.*, 32(3):3–9, 2009.
3. Marlon Dumas. On the convergence of data and process engineering. In *Proc. of the 15th International Conference on Advances in Databases and Information Systems (ADBIS), Vienna, Austria*, pages 19–26. Springer, September 2011.
4. Thomas Erl. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall, 2005.
5. P. Feldman and D. Miller. Entity model clustering: Structuring a data model by abstraction. *The Computer Journal*, 29(4):348360, 1986.
6. Richard Hull, Elio Damaggio, Riccardo De Masellis, Fabiana Fournier, Manmohan Gupta, Fenno Terry Heath, Stacy Hobson, Mark H. Linehan, Sridhar Maradugu, Anil Nigam, Piyawadee Noi Sukaviriya, and Roman Vaculín. Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In *Proceedings of the Fifth ACM International Conference on Distributed Event-Based Systems (DEBS), New York, NY, USA*, pages 51–62. ACM, July 2011.
7. Thomas Kohlborn, Axel Korthaus, Taizan Chan, and Michael Rosemann. Identification and analysis of business and software services - a consolidated approach. *IEEE Transactions on Services Computing*, 2(1):50–64, 2009.
8. Vera Künzle and Manfred Reichert. Philharmonicflows: towards a framework for object-aware process management. *Journal of Software Maintenance*, 23(4):205–244, 2011.
9. Guy Redding, Marlon Dumas, Arthur H. M. ter Hofstede, and Adrian Iordachescu. A flexible, object-centric approach for business process modelling. *Service Oriented Computing and Applications*, 4(3):191–201, 2010.