What About Database-centric Enterprise Application Integration?

Daniel Ritter

HANA Platform, SAP AG
Dietmar-Hopp-Allee 16, 69190 Walldorf, Germany
daniel.ritter@sap.com

Abstract. The focus on “big data” and the emergence of hybrid Online Transaction Processing/Online Analytical Processing systems in the database community creates new opportunities for (business) application vendors. More and more business logic is “pushed" to the database, i. e., close to the application data, for faster and more efficient processing, while avoiding unnecessary data shipments.
In this position paper, we argue that *Enterprise Application Integration* should engage in a liaison with the recent data-processing advances, especially for supporting the integration of applications running in the database with remote applications.

Keywords: Enterprise Application Integration, Relational Database.

1 Introduction and Position

The recent advances within the database research community–not limited to hybrid transactional and analytical systems (e. g., [5,9])–bring applications together on one platform: the database. For less data shipment and more efficient processing, (business) application logic is “pushed" to the databases by translating it to standard SQL, PL/SQL and an increasing number of additional libraries and programming languages [1,2]. The databases of complete business application suites[1] are not only used for “bookkeeping" anymore, but transform to application platforms. The conventional application systems remain the presentation layer, while the control and data flows move closer to the databases.

These applications require message-based integration. Some of the data and message endpoints like *Data Stream and Complex Event Processing* (CEP), for *Information Flow Processing* [4], and *ETL* (e. g., *Microsoft SQL Server Integration Services*) already operate on database level. While these data endpoints care about high-performance inbound data loading, (message) protocol and format-level transformations (e. g., JSON, XML to relational model), data cleansing and storage to database tables, standard integration capabilities like routing, mapping, and guaranteed delivery are left for EAI systems, which are still implemented on application server level. When additionally considering the subsumption premises

[1] For example, the SAP Business Suite on HANA: https://www.suiteonhana.com

of “Too much Middleware" [11] (i. e., discussing the costs/benefits of a separate EAI system) and the requirements to middleware systems that are already well-covered by most database systems (e. g., scalability, data consistency/transaction processing, user management, high availability), we target the question whether message-based integration is viable from a database perspective. The required system shall realize EAI solely using standard relational processing (e. g., SQL, PL/SQL) for an efficient evaluation of integration semantics. We address the following research questions, discussed subsequently: (1) “Can integration logic (e. g., mapping, routing, aggregation [7]) be “pushed" into the database completely?", (2) “For which *Integration Scenarios* does that bring which benefits?", and (3) “What are its advantages and disadvantages?"

2 The Database as Integration Middleware

We found recent evidence to our position in the areas of declarative message processing in XQuery/XML-DBMS (e. g., [10,3]) and *Publish/Subscribe* in relational databases [6]. Since this work considers only a small subset of integration semantics, requires massive extensions to standard SQL and most business application data is stored in relational databases, these approaches only support its overall position. Subsequently, we briefly sketch the most relevant aspects, namely *Integration Cases* and *Common EAI Scenarios.*

Table 1. Relevant *Integration Cases* considered for database and integration systems (Middlew.) based on control and data flow. Relevant case for this paper is **highlighted**.

Cases	Data Flow	Control Flow	Comment
Case 1	Middlew.	Middlew.	no persistent storage of messages; synchronous messaging
Case 2	**Database**	**Database**	**Database with Protocol Adapters (no Middlew.)**
Case 3	Database	Middlew.	Middlew. controls execution on Database and its Adapters
Case 4	Middlew./Database	Middlew./Database	Shared control and data flow
Case 5	Middlew./Database	Middlew.	Middlew. controls shared data processing
Case 6	Middlew./Database	Database	Database controls shared data flow

Integration Cases For a systematic definition of the term “database-centric EAI" we distinguished six different and relevant integration cases between the poles of control and data flow for database and integration systems. These cases are listed in Table 1. We focus on *Case 2*, in which the database handles the control and data flow execution exclusively, while no additional application tier for the middleware system (short “Middlew.") is required. In general, EAI systems consist of protocol adapters (i. e., message endpoints) and integration logic. The endpoints are assumed to be available on database-level (e. g., CEP, ETL on inbound and messaging [6] on inbound and outbound site). The integration logic is expressed as standard database artifacts like SQL, PL/SQL. For comparison, we defined *Case 1* as integration system without storage (e. g., no asynchronous messaging possible) and *Case 5* refers to current EAI systems with persistent message storage (e. g., Java Messaging Service (JMS)[2] based processing) and control over message processing and data flows.

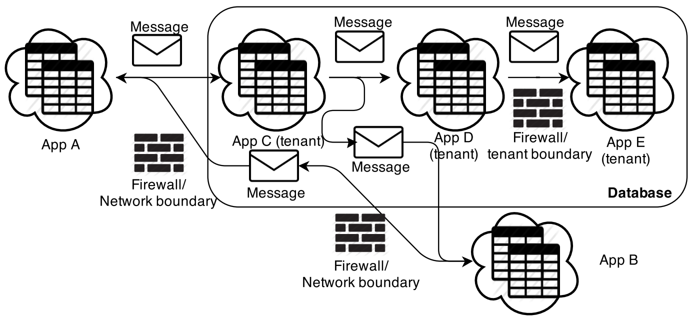

Fig. 1. Common EAI Scenarios: C1 (*App C* to *App D*), C2 (in: *App A* to *App C*, out: *App C* to *App B*) and C3 (*App A* to *App B* via *database*). In case of cross-tenant communication (*App D* to *App E*) we assume equivalence to C2 (outbound/inbound).

Common EAI Scenarios Figure 1 shows an overview of common EAI scenarios that are combined to categories *C1-3*. Category *C1* (local/local: from *App C* to *App D*) refers to the integration of a sender and a receiver application residing in the same application system, sharing the same database, but not the schema (e. g., as in business suites). Category *C2* (local/external: inbound processing from *App A* to *App C*; outbound processing from *App C* to *App B*) covers the cases, in which an application on the database (sender or receiver) needs to access or receive external data (e. g., access remote applications). Finally Category *C3* (external/external: from (*App A* to *App B*) features sender and receiver applications residing in different application systems, while communication is mediated by a dedicated “integration database system" (i. e., corresponds to the classical middleware case). The “multi-tenant database" case from [8] is depicted by the communication from *App D* to *App E*, which requires an even stricter tenant and schema separation that eventually translates to C2. As example for a C2 scenario, we selected the SAP ERP *Convergent Invoicing* (FI-CA) use case. For the invoicing process, high-volume billable items (up to 500k per second–approx. 4 billion messages per day) wrapped as messages (e. g., one iTunes song or telephone call decomposes to records for customer, vendor, author etc) are created and sent by the *Convergent Charging* application. The messages have to be validated, enriched with master data, filtered and aggregated according to multi-dimensional and potentially customer specific criteria, before legally binding documents are created. JMS messaging could queue that amount of messages[3], however, cannot handle the integration logic.

[2] Java Messaging Service: http://www.jcp.org/en/jsr/detail?id=914
[3] JMS benchmark: http://www.spec.org/jms2007/

3 Discussion

The research questions formulated in Section 1 (i. e., (1)–(3)) target the feasibility and viability of a database-only integration approach (see Case 2, Section 2). The question about integration logic "push-down" to the database system (1) was approached experimentally by a prototypical implementation. The evaluation showed short-comings in terms of language expressiveness (e. g., timed-aggregations are not possible), the need for a scheduled pipeline processing, when a (transactional) decoupling between sender and receiver is required, and a small latency penalty when mixing SQL and PL/SQL processing. Besides that, the processing of bulks of messages showed results for throughput comparable to the requirements, e. g., of the FI-CA scenario, thus leading to the question about viable scenarios (2). The natural category for database-centric processing seems to be C1 for database local, schema-to-schema integration. However, this is currently completely covered by "shared-schema" integration, although it could untie the tight application coupling. The support of database adapters with pre-/post-protocol conversion integration logic processing C2 seems most promising (see FI-CA scenario). Category C3 is the domain of current EAI systems and only seems to show benefits for scenarios with qualities that require frequent persistent message storage and/or massive data-bound computations for database-only integration. The benefits and additional operational qualities (3) are stable asynchronous and transactional message processing, portable database code (for standard SQL logic only), interoperability through database protocol adapters, and less frequent and expensive data format conversions.

References

1. C. Binnig, N. May, and T. Mindnich. SQLScript: Efficiently Analyzing Big Enterprise Data in SAP HANA. In *BTW*, pages 363–382, 2013.
2. C. Binnig, R. Rehrmann, F. Faerber, and R. Riewe. Funsql: it is time to make sql functional. In *EDBT/ICDT Workshops*, pages 41–46, 2012.
3. A. Böhm, C.-C. Kanne, and G. Moerkotte. Demaq: A foundation for declarative xml message processing. In *CIDR*, pages 33–43, 2007.
4. G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15, 2012.
5. F. Färber, N. May, W. Lehner, P. Große, I. Müller, H. Rauhe, and J. Dees. The SAP HANA Database – An Architecture Overview. *IEEE Data Eng. Bull.*, 35(1):28–33, 2012.
6. D. Gawlick and S. Mishra. Information sharing with the oracle database. In *DEBS*, 2003.
7. G. Hohpe and B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
8. D. Jacobs and S. Aulbach. Ruminations on multi-tenant databases. In *BTW*, pages 514–521, 2007.
9. A. Kemper and T. Neumann. One size fits all, again! the architecture of the hybrid oltp&olap database management system hyper. In *BIRTE*, pages 7–23, 2010.
10. N. Onose and J. Siméon. Xquery at your web service. In *WWW*, pages 603–611, 2004.
11. M. Stonebraker. Too much middleware. *SIGMOD Record*, 31(1):97–106, 2002.