Drafting a General Framework for Systems Integration Delivery

Bartosz Marcinkowski
University of Gdansk, Gdansk, Poland *bmarc@ug.edu.pl*

Bartlomiej Gawin
University of Gdansk, Gdansk, Poland *bartlomiej.gawin@ug.edu.pl*

Abstract

In IT ecosystems that imply the exchange of data with systems on the customer or business partner side, it is oftentimes problematic or even borderline impossible to predict what external systems shall eventually be covered by integration and what technological solutions should be employed. Based on the feedback obtained in a series of interviews, we lay a foundation for building a framework for developing system integration. The framework enhances the research background in terms of technological specificity and constraints behind developing projects of this nature.

Keywords: System Integration, Software Engineering, Third-Party Systems, Framework.

1. Introduction

Information systems being developed and deployed across companies are not isolated islands. In a contemporary world that is heavily saturated with information technologies, both ready-made and custom IT solutions are embedded in some existing technological settings. During the analysis and design stage of such solutions, it is rarely thought of that these systems shall eventually be integrated within a specific context. Granted, vendors of popular software design and document Application Programming Interfaces (APIs) to facilitate the most common integration needs. These APIs are capable of receiving and providing data. The usability of APIs in general cannot be taken for granted [1]. As a result, it oftentimes turns out that the integration process considered in business terms is inextricably related to linked to developing a new utility software, which entails time, costs, and certain risks. The integrated IT solutions can be rolled out as brand-new ones, which makes adaptation and testing manageable. Alas, integration needs can also follow building blocks that have worked for some time and are being replaced, rehauled, or upscaled – not to mention that it often involves multiple customer-side IT solutions. Therefore, integration-oriented projects pose a number of technological challenges [2]. One has to consider, among others, issues related to data exchange protocols, transfer reliability issues, and the very formats that data is exchanged in [3]. In this paper, we employ preliminary knowledge from a set of 14 semi-structured interviews to sort out the research background on the types of system integration, the development process of system integration, and the strictly technological setting of such a process.

2. Technological Components of the Framework

The study shows that, as a general rule, one should use the integration types over which one has control in the long term. On the other hand, the peculiarity of building integration solutions with business partners is that the choice cannot always be brought down to a solution best technologically (Table 1). Reasons for this include the limitations of external systems or the inability to directly interfere with them in many situations. That said, APIs at the library level, as well as Web APIs, were mentioned most often. The latter included various Representational State Transfer (REST), Simple Object Access Protocol (SOAP), and Resource Description Framework (RDF) implementations. In particular, REST APIs got a lot of praise for being generally well-documented. However, some drawbacks were also highlighted (see Table 1). Aside from APIs, respondents

specifically singled out email integration, file integration, intermediary/shared databases, queue managers, data buses as well as crawling integration. In the case of the former, a target system or middleware receives specific data from an email, processes it, and carries out a business process or action as a result. As far as file integration and shared databases are concerned, the mechanism is simple: one application generates a dataset, and the other imports it. The latter case in particular comes with a risk of mass import of incorrect data due to failure. Queue managers and data buses proved convenient in highly distributed application ecosystems. Datasets make their way to the bus, with the format of data being of secondary importance. Individual integrated applications are simply plugged into the bus. Crawling integration tended to be treated as a necessary evil. This was due to the potential instability of a solution that simulates the operation of a web browser. Legacy implementations favored PhantomJS – a scriptable browser that came with no GUI. Headless Chrome may serve as a faster and more modern alternative. When taking advantage of crawling, page rendering is accomplished in the background.

Table 1. Types of system integration.

Issue	Respondents' feedback
Decision factors	• "*sometimes an integration that is, from a coding point of view, not such a neat or refined thing gives a big business gain, fits into economic dimensions*" • "*predictability has a very big impact on the choice of integration; since crawling integration relies on the appearance of a specific website, it is necessarily burdened with a lot of variabilities; sometimes all it takes is for someone to change the settings in the user panel – and that will already affect the integration; and an API, given that it is well-designed and versioned, you can assume that it is practically stable*" • "*should we bind applications directly, especially in the case of synchronous activities, this may lead to visible non-responsiveness periods of bound systems; implementation of such integration sometimes requires, for example, adding additional events in the source system (e.g., «a note was added»)*"
APIs	• "*if the system has an imprecisely described API, or these endpoints are not typical ones (and we can only rely on Hypertext Transfer Protocol POST requests) – well, we kind of stimulate the action of the browser and send an authenticated form via HTTP*" • "[counterintuitively, this is not necessarily the most pleasant interface]; *whereas in our project it was self-describing and well-defined, it was the eXtensible Markup Language that served as the exchange format; it has a substantial overhead; there is a higher entry barrier, a greater challenge to implement it well*"
Email integration	• "*we receive outgoing email from customer systems (in the form of notifications, e.g.: «a new request was created», «a note was added» to a specific request/ticket, «the status of the request changed»); we have such a bulk mailbox: mails are parsed, we extract vital data from their contents, and we forward it to the API of the target system*" • "[the case may be more complex, however]: *an email from the customer system might not have enough data for such a request to be created on our side; when data gaps are detected, a message that requests data supplementation is sent to the queue system; the data is populated via crawling; once the necessary data has been assembled, a «data collected» message is sent; this message is consumed by the email integration once again, and the latter asynchronously completes the interrupted process*"
File integration	• "*whether it is manually added after exporting from other systems, or shared via the File Transfer Protocol, or cloud solutions; primitive, yet nevertheless integrates systems, exchanges information between them*" • "*as part of the workflow, an XML file is created, which is not sent via SOAP, but is placed on disk; it goes to the right folder, and the business partner's system at some interval consumes it and proceeds further*"
Queue managers	• "*there is potential for scalability here because you pay once for implementing a message broker, and you can tack on more applications that use the same messages; these systems do not necessarily need to know about each other, as long as the formats are maintained*" • "*you need to take care of RabbitMQ's redundancy since there are problems when you fail to connect to the queue manager, and there is no internal buffer within the application*"
Crawling integration	• "*it behaves as if a normal user enters, and is triggered by an automaton – along the lines of automatic Selenium tests; we can select any elements on a page, click on them, switch between pages, and so on; under normal circumstances, the external system should not even detect this*" • "*such integration was simply motivated by economic considerations; the third-party company that developed the system on the customer side wanted a serious amount of money for fielding an API*"

Distinctively, both the final decisions on the choice of integration types and the very rationale for its implementation were far more often business-driven rather than technology-driven. Strictly technological drivers were limited to internal applications or basing the architecture on microservices (in order to ensure decoupling, redundancy, and scalability). One of the key characteristics of streamlining the performance was the suppression of errors associated with manual data entry across systems ("*manual work is tedious, here you have to log in, there log out... this affects the comfort of working with these systems as well as the overall erroneousness*"; "*transcribing data from one IT solution to another asks for inaccuracies, gaps, and misrepresentations; if we map one system to another, there is virtually no such possibility – of course, as long as we do our*

programming right"). It is not inconsequential that once a certain volume of data exchange is reached, it becomes unmanageable for manual processing. The lack of even basic business process automation affects response times or Service-Level Agreements (SLAs).

Generally speaking, integration proceeds like a casual IT project. Here, too, one has to go through planning, collect system requirements, design solutions, and so on. That said, there are different focus points. Past projects were run both in accordance with agile practices [4] and classically (waterfall approach). Respondents reiterated the more intensive use of analysis-level artifacts as well as the need for mapping data and events between systems soundly (Table 2). An extensive number of individual processes' scenarios and the tendency to focus too much on the main scenario add to the difficulty. Such documentation proved to be essential in the future for rapid reaction both during the stabilization phase and subsequent maintenance. Usual analytical and design activity tended to be complemented by elaborating Proofs of Concept for connecting to given systems. This facilitated detecting technical and business problems in advance (e.g., something cannot be mapped, something presents a difficulty when confronted with technology at hand, or key information is missing). The decision to opt for a specific integration type proved to be a significant milestone. As far as the stabilization phase and events during subsequent maintenance are concerned, rigid deadlines, and a narrow timeframe for completing workflows were highlighted.

Table 2. The development process of system integration.

Issue	Respondents' feedback
Process	• "*the choice* [of methods and techniques] *depends on how the company works, on what terms one generally collaborates with a given client, what the specific needs are, whether it is an emergency thing, whether there is comfort* [in terms of time] *to fully analyze, understand the business, generate documentation – and only then move to implementation*" • "*intensive use of prototyping* [stands out regarding such projects]; *this many a time helped a lot – you can tell the business already at very preliminary stages that indeed, it is possible (or just the opposite) and build target solutions on the basis of such prototype*"
Analysis	• "*the integration with the ERP-class system was too focused on how a particular message would be built and encapsulated – and not enough on what data was going through, what should be mapped on what, what that meant later on business-wise, what ERP-side structures would be responsible for supporting the practices on in our local system in order to later extract, for instance, some financial or analytical data from that; as a result, in the middle of the development work itself, there was re-analysis, re-evaluation of data sources; it was apparent that this was deficient before*" • "*we need to know what a user is entering and what is expected as a consequence in the system being integrated (e.g., we set a status locally, and then in the XXX system, the status is supposed to follow suit and a notification pops up); there may be user stories or the Unified Modeling Language across the documentation – the business case is what invariably comes to the fore*" • "*that is, rather than user stories, it is more like system stories; if someone does not think this through, it becomes s a terrible mess; a lot of changing, meddling; do need detailed business process models and their system interpretation*" • "*it would be wise to additionally describe the technical flows – what triggers individual transitions, what information goes through each system, where the endpoints are, what the responses are, what happens in critical situations; this facilitates later maintenance, knowledge propagation*" • "*it was convenient that several integration choices were on the table; and with the availability of a trial version* [of the system to be integrated], *we were able to assess the suitability of the various options; this is where the fact that the team was narrowed down got in the way a bit; for such an extensive integration, there should be people who are at odds with certain solutions, to take advantage of their expertise*" • "*based on the experience of the earlier integration of the financial and accounting system* [of national scope], *the basic criterion was that this process should be asynchronous and well protected against data loss in case of temporary unavailability of the infrastructure; these circumstances allowed us to design the integration in such a way that we would not be constrained in case we needed to adapt to changes or a necessity emerged to add support for more endpoints*"
Closing stages	• "*stabilization is a bit more challenging, because it is, after all, a living system (and let us not forget that an external one); should something go sideways, some business process is interrupted and you would not even blink before there was time tension, pressure from the business community to execute quickly at the expense of simplification*" • "*it was nerve-wracking; combined with the fact that there were over a hundred thousand integration messages, the stabilization protracted; this was not an isolated incident*" • "*the maintenance is basically OK until an external system goes through an unexpected modification; this is where monitoring comes into play*"

Consideration of the aforementioned factors made it possible to identify major components of the framework and to elaborate on strictly technological issues (Fig. 1).

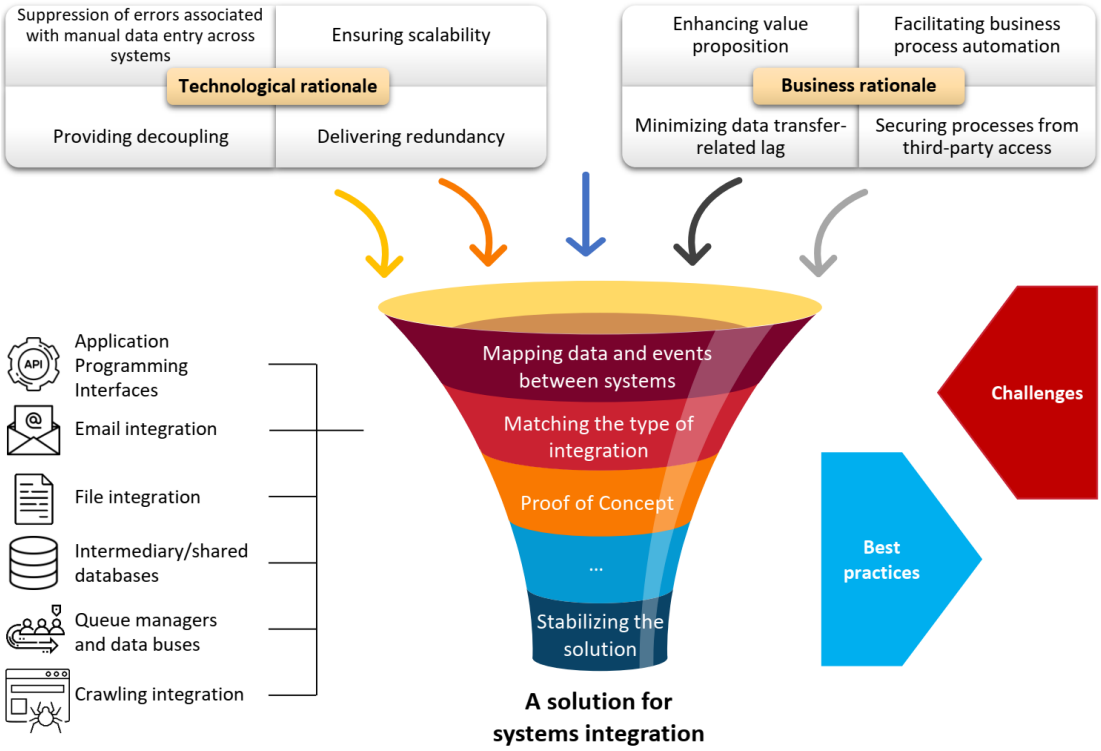

Fig. 1. General framework for systems integration delivery.

3. Future Work

Technological factors cannot be ignored in the implementation of any kind of ambitious integration process. This paper organized and expanded the list of the core technological stacks behind providing integration originally compiled by Hohpe & Woolf and often referenced ever since [5]. That said, adequate process design and consideration of the specific technological factors of integration projects is at best a necessary and not a sufficient condition for the success of such projects. Carrying on the work commenced in this paper, we shall focus on the business aspect of integration projects – scrutinizing, in particular, the challenges, success factors, and best practices of building integration of IT solutions. By factoring in these additional dimensions We hope to provide both management- and specialist-level staff with a plane of reference for seamlessly handling integration-centric projects.

References

1. Venigalla, A. S. M., Chimalakonda, S.: On the Comprehension of Application Programming Interface Usability in Game Engines. Software: Practice and Experience, 51(8), 1728-1744 (2021).
2. Du, L., Duan, C., Liu, S., He, W.: Research on Service Bus for Distributed Real-time Control Systems. In: 6th IEEE Joint International Information Technology and Artificial Intelligence Conference, pp. 401–405. IEEE, New Jersey, NJ (2011).
3. He, W., Da Xu, L.: Integration of Distributed Enterprise Applications: a Survey. IEEE Transactions on Industrial Informatics 10(1), 35–42 (2014).
4. Joskowski, A., Przybyłek, A., Marcinkowski, B.: Scaling Scrum with a Customized Nexus Framework: A Report From a Joint Industry-Academia Research Project. Software: Practice and Experience, 1–18 (2023).
5. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley, Boston, MA (2004).