Deconstructing the ‘decoupling’ in integration architecture

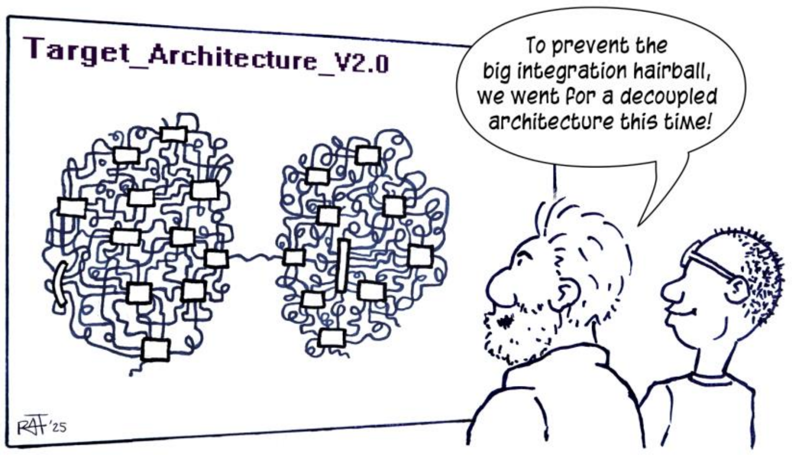
Dr. Ralph Foorthuis

LinkedIn article - January 15th 2025

Summary: There are multiple architectural approaches for decoupling, and some of their characteristics are complementary while others are mutually exclusive. There are multiple ways to decrease dependencies, but some dependencies are actually the ones we get value from. Make sure you know the trade-offs!

Introduction

Although you can argue we only have a handful of patterns for sharing data in our IT landscape, application integration is a highly complex and rich domain. The realization of each integration demands a thorough understanding of the requirements and context, as well as a proper assessment of alternatives and their pros and cons. This holds even more if we take decoupling into account.


Target_Architecture_V2.0
To prevent the big integration hairball, we went for a decoupled architecture this time!
RAF '25

The many faces of decoupling

‘Decoupling’ and the more modest ‘loose coupling’ are all too often used terms in discussions on architecture and integration, usually put forward as a self-evident magical bullet. But effective decoupling is in fact a multifaceted beast, strongly situation-dependent and often confronting you with many trade-offs. Important factors in this context are – amongst others – the number of involved systems, time performance, timeliness, robustness, frequency and volumes of the data exchange, push versus pull, reusability, security, (a)synchronicity, scalability, transparency, (non)standardized data models, duplication of data and systems, complexity, version management, amount of work, deadlines, and many other aspects that relate to loose versus tight coupling in intricate ways.

There are various approaches for ‘decoupling’ and many of the different pros and cons they come with are mutually exclusive. We will walk through some popular approaches below. A first one is designing your applications and your entire landscape in a modular fashion. This means for example implementing micro services or developing your applications in a layered structure. These modules need to be tightly coupled from an internal perspective but work relatively independently from each other. Modularization is a popular approach to manage complexity, scale applications, and separate frontends and backends so as to gain flexibility and maintainability. However, be aware that it actually also introduces complexity due to the need for new external interfaces, internal variables, functionality and cross-module orchestration.

The need for such well-managed interfaces may be reason to also employ another approach, namely using an *enterprise-wide canonical data model* as a blueprint for data sharing formats. Such a data model allows creating standardized, understandable datasets and interfaces that can be used by many stakeholders. This prevents that you end up with an unmanageable hairball architecture and an unnecessary explosion of point-to-point integration designs and future maintenance tasks. Using the data model for physical implementations decouples receivers from changes in the source’s technical model, increases shared data understanding, and should make it easier to ‘plug’ systems in and out of the landscape. At the same time, however, as more and more systems make use of a reusable event or API the dependencies across the landscape also increase.

This bring us to decoupling approaches that are directly related to middleware. An *API gateway* decouples the external and internal worlds, and as such hides and protects backend components from malicious external actors. It may also offer some transformation functionality in order to make your applications less dependent on each other’s protocols, interfaces and addresses. However, the gateway is typically used for obtaining the most recent data via request-response interactions. This requires that the source and target are simultaneously online – and thus remain very coupled. On the other hand, *(event) message brokers* will retain the data messages until the subscribed target systems have picked them up and therefore bring such technical decoupling. However, the source doesn’t know from this integration whether the targets successfully received the message, so if this needs to be transparent then additional integrations (and dependencies) are required. Depending on the used middleware it may not be easy to do data transformations and thus may come with data schema dependencies.

More approaches can be mentioned, such as clearly *separating operational and analytical data.* This prevents that systems for reporting and advanced analytics get unnecessarily dragged into highly operational business processes that require very different technical optimizations and service levels. Each of the decoupling approaches discussed above brings its own benefits and drawbacks, many of which are mutually exclusive. After all, you cannot have real-time access to another system’s data without a direct dependency. It is therefore crucial to be aware of the trade-offs in order to choose the best option.

Trade-offs

Let’s look in more detail at some examples of trade-offs involved in decisions on integration. Faster performance and scalability are beneficial for users and business, but often come with various forms of duplication as well as increased cost, overhead and data inconsistency. Standardization and reusability, e.g. using enterprise data models, work well for robust and efficient integrations in the future, but come with longer implementation times in the here and now. Keeping multiple versions of a generic API message simultaneously in production offers backwards compatibility and prevents a big bang transition for the many systems involved, but also leads to more complexity in the source or middleware.

Aren't there any no-regret practices then? Sure there are. If you need to share updates with multiple target systems directly after they occur, then an (event) message-based pub/sub approach is a good way to prevent a large number of individual calls to the source and to ensure that the source and targets do not need to be simultaneously online. And if security, standardization and integration monitoring at the landscape level are important, it is a good practice to have your integrations by default be mediated by an API gateway.

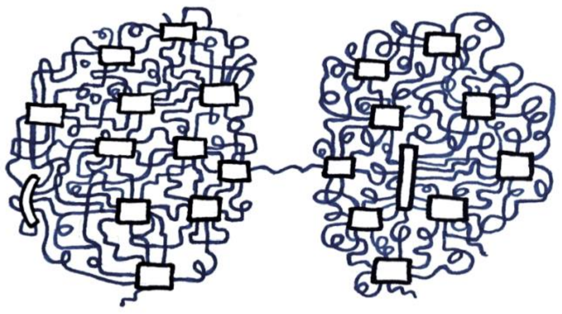
However, the "if" and "by default" already illustrate the conditionality, and it surely doesn't stop there. In the API gateway example it may also be crucial to deliver top performance in which every millisecond counts, making you decide not to put the gateway in between after all. Moreover, the gateway is an additional component that not only allows decoupling between domains, but also introduces another coupling – another dependency and point of failure – in the end-to-end data flows.

In the message pub/sub example other crucial factors are how many systems manage the original data, the size of the messages, whether it is important that the source is (immediately) aware of the success of the integration, and to what extend the integration needs to be guaranteed. Depending on these factors one or the other integration solution may be more fitting, bringing in its own pros and cons.

Optimal

A perfect setup usually does not exist and we often arrive at optimal (de)coupling because in our connected world the underlying dependencies simply exist and need to be dealt with. These dependencies manifest themselves in many different ways because they pertain to all kinds of aspects: business processes and functionality, technical elements, data models and data values, implementation deadlines, et cetera. And they will often confront you simultaneously because a dependency between different business processes will per definition come with technical dependencies if some level of automation is desired.

An important goal of decoupling is to minimize the impact of both unplanned disturbances and planned changes, knowing that the many dependencies in our current IT landscapes will unavoidably confront us with complex inter-system impacts. Fortunately, robust business applications, powerful integration platforms, high-quality standardized data models, preventative checks, skilled professionals, and proper management processes help to reduce those undesired consequences. But… in many cases we will need to balance the pros and cons to arrive at the optimal integration.



Good to know: wrong decisions on integration still result in decoupling. However, it will be a decoupling between your business goals and IT support, in other words a ‘business/IT misalignment’! This is of course what we aim to prevent, as this may lead for example to a daily batch transfer while the business context actually demands real-time insights into the current status of the sales order and other data objects. Or you may end up with a highly standardized integration that proves not to be scalable and offers poor performance during busy hours.

Takeaway

One decoupling approach is not the other, and some of their characteristics are complementary while others are mutually exclusive. There are multiple ways to decrease dependencies, but some dependencies are actually the ones we get value from! There is no silver bullet, so for each integration you should make sure that you have a good understanding of the situation and of the pros and cons of the options. And to avoid disappointment, ensure that your stakeholders accept the limitations of the chosen solution!