A Cloud Service Integration Architecture for the Hospitality Sector

Paulo A. Melo
CeBER - Centre for Business and Economics Research / University of Coimbra and INESC Coimbra
Coimbra, Portugal *pmelo@fe.uc.pt*

Paulo Rupino Cunha
CISUC, Department of Informatics Engineering, University of Coimbra
Coimbra, Portugal *rupino@dei.uc.pt*

Ricardo Amaro
Instituto Pedro Nunes
Coimbra, Portugal *rjamaro@ipn.pt*

Carlos Lopes
Instituto Pedro Nunes
Coimbra, Portugal *clopes@ipn.pt*

Ricardo Madeira
Hotelcracy
Coimbra, Portugal *ricardo-madeira@hotelcracy.com*

Pedro Miguel Antunes
Hotelcracy
Coimbra, Portugal *mantunes@hotelcracy.com*

Abstract

We describe the design and implementation of an innovative software platform that enables hoteliers to configure custom solutions to manage their business, by building on multiple existing Software-as-a-Service (SaaS) solutions. A flexible cloud integration mechanism uses "drivers" to communicate with various SaaS via their existing APIs. Selection of the solutions is made easy by the inclusion of a Marketplace, and the user experience is made seamless by an homogenous user interface (UI) that orchestrates the calls to the heterogenous underlying logic provided by multiple vendors. The resulting platform is, itself, offered as a SaaS, whose functionality is highly adaptable depending on the chosen integrations. These can be reconfigured at any time, thus avoiding lock-in to any particular one, and without incurring in the costs associated with retraining and learning curves of end-users. The core cloud integration architecture of the platform is generic, so it can be adapted for other domains.

Keywords: SaaS, Cloud integration, Hospitality.

1. Introduction

Outside of the well-known (and large) hotel chains, the hospitality accommodation sector is mostly comprised of independent hotels and lodgings, particularly in Europe, where about two-thirds of rooms are supplied by independent firms; but even in the US, they account for about 30% of rooms [17]. Independent companies are those not affiliated with chains or brands imposing standardized service and/or processes. They are characterized by their diversity, sporting wide differences in areas such as price, size, and organizational

structure, while simultaneously having to overcome some competitive disadvantages (namely in pricing power) relative to the larger chains [3, 11]. Due to their heterogeneity, coupled with small size on average, most independent units have limitations in resources and technical knowledge for information technologies (IT) in comparison with the hotel chains.

That said, in the last 20 years information and communication technologies have gained enormous importance in the management of hospitality facilities [13], namely with the success of Online Travel Agencies (e.g. Expedia, Booking.com) and of sharing economy hospitality platforms (e.g. Airbnb), forcing new business models into the industry as a whole [21, 22].

Although integrated IT tools exist for small to medium-sized hospitality operators, they usually take the form Property Management Systems (PMS) [5], which support the operational tasks, including room management and guest check-in and checkout. Traditionally supported by in-house IT staff, particularly in large chains, nowadays such systems are increasingly being delivered via cloud-based Software-as-a-Service (SaaS) solutions. Given their ERP-like nature, PMSs usually constrain the integration with solutions from other vendors, leading to high costs and lock-in. They often also assume or encourage one-size-fits-all business models, which may be inadequate for certain hospitality businesses. Finally, these PMSs usually evolve according to development preferences of the vendor, sometimes missing advances and changes in the market that create new needs. Recent examples include the requirement of hospitality companies to have a strong digital presence, to ensure daily competitiveness of prices, and to respond quickly to customer's needs [4]. These have led to the development of single or narrower-purpose tools, whose use quickly became a critical success factor for the industry. Such tools include: the hotel website, the channel manager (to support different online distribution channels), the revenue management system (to define adequate rates for the advertised services in face of changing conditions), the rate shopper (to monitor online rates for similar services), the reputation management system (to manage the online presence and advertising), and a billing tool (to accept payments and consolidate the revenue channels). Table 1 presents a summary of 15 tool categories used nowadays in the hospitality industry. For each of these categories, Cloud Software-as-a-Service (SaaS) solutions are frequently available on a pay-as-you-go model. To address their various needs, hotels must often subscribe to a portfolio of such tools, which, in turn, makes them harder to manage and/or integrate into the overall organizational information system.

The adoption of several of the SaaS services in the above categories by smaller independent hotels and lodgings can become a time-consuming and expensive endeavor, due to factors such as the need for initial and ongoing analysis and selection of which ones to retain; the need for user training whenever a new service is contracted; and the effort to integrate them with the organization's current IT portfolio. Frequently, the hotel's own staff selects the solutions, often without adequate technical knowledge, resulting in disorganized, overlapping collections, poorly aligned with the organization's business processes and goals, often leading to duplication of services, tasks, and data. The latter issue may, in turn, beget errors, which affect the hotel bottom-line, impacting its competitiveness. In any case, mismatched overlapping duties imply misallocation of human resources that could be focused instead on ensuring customers a good stay.

In this paper, we describe the design of an innovative software platform aimed at solving the above challenges for small and medium-sized hotels. A Software-as-a-Service solution itself, it works by integrating other existing Software-as-a-Service best-of-breeds, selected from a marketplace, to create a customized solution offered via a homogenous user interface.

The problem of integrating cloud-based services to provide a single point of control has been under research for quite some time. For an early description of the requirements for SaaS integration platforms, see [20]. However, although such integration has long been considered a need for the successful adoption of cloud services [9], that is far from easy in many application domains.

In recent years, the act of providing generic integration-Platforms-as-a-Service (IPaaS) has been gathering some traction [8]. However, as far as the authors know the support for the hospitality sector in these generic platforms is very limited, which is not unexpected, since most publicly available platforms target high-volume services due to cost/benefit analysis.

Table 1. Hospitality Service Tool Categories (partially adapted from [2])

Tool Category	Description
Access management systems	Manage remote access to properties, allowing the opening of doors to rooms and houses via the Internet, generate temporary access, and know who accessed, avoiding the need for a physical presence of the owner/manager.
Billing and payments	Manage the tracking of products and services provided, and invoice them to customers. Automate time-consuming tasks such as invoice preparation.
Booking Engine	Allow the hotel to provide its customers with a way to reserve directly on its website, so it doesn't have to pay sales commissions to third parties on those reservations.
Channel Manager	Provide rooms through various online distribution channels, such as booking websites or tour operators; manage reservations made at different online locations; manage prices and room availability.
Customer Relationship Management (CRM)	Support tasks including collection and analysis of customer information; manage complaints; develop and execute customer-facing advertising campaigns.
Finance management	Plan, analysis, and control of accounts, allowing the managers to visualize the financial situation of the organization.
Human Resources Management	Manage and process information pertaining to the human resources of an organization, for example, allowing scheduling work hours and employee communication.
Maintenance operations	Manage the infrastructures and equipment maintenance, such as fault control and repair interventions in buildings and equipment.
Mobile Applications	Digital tour guides and valet/front-desk staff, that provide relevant information to customers and non-customers such as directions to accommodation or points of interest in the vicinity.
Point-of-Sale System (POS)	Replace for the cash register and for transactions carried out inside the facility, usually in the bar and restaurant services.
Property Management System (PMS)	Manage the hotel's central operations, such as guest check-in and checkout; manage guest profiles; front and back office functions; audit; manage information relating to rooms and cleaning services.
Rate Shopper	Collect and analyze competitive prices; such information can be used by a Revenue Management System to maximize profit.
Reputation Manager	Automate the analysis of reviews shared by guests on social networks, so that the hotel can act more readily on possible identified problems.
Revenue Management System	Control the hotel's inventory (quantities of rooms and respective prices), with the aim of obtaining the maximum profit, e.g. by applying dynamic rates depending on the occupancy of the room.
Virtual Automatic Payment Terminal	Payment systems allowing electronic payments to be made using bank cards without the need for a physical Payment Terminal.

Several generic architectures have also been proposed in the literature to allow for service reusability and limit lock-in. Notable among those are the Cloud Computing Open Architecture – CCOA [24] and the Service-Oriented Cloud Computing Architecture – SOCCA [23]. However, none of these has yet gained an implemented base that would allow for creating a solution based on these them. Therefore, our proposed architecture tries to create a less generic but more directly implementable solution.

Finally, studies have also been made on existing cloud integration patterns, like [14]. If placed within the framework of this study, our platform would show some characteristics of a SaaS broker, which would fit the requirements of a domain-limited approach with no need for large SaaS-to-SaaS communication. However, the inclusion of a local data-repository creates a variation of the simpler model.

The remainder of this paper is organized as follows: in Section 2 we detail the platform architecture, down to its components, explaining how it was shaped by domain concerns

and use cases. Section 3 describes some tests performed to evaluate the architecture. Finally, Section 4 closes with conclusions and future work.

2. Platform Architecture

One key requirement for the software to be developed was that it should be offered as an off-the-shelf SaaS that would integrate several other commercial domain-specific SaaS solutions that the hoteliers would select from a marketplace. Additionally, it should enable that such solutions can be swapped out and their data migrated to the new ones, thus avoiding vendor lock-in. Furthermore, it should provide a consistent user interface across the heterogeneous integrated SaaS services, enabling for a smooth experience and reduced training costs and learning curve. It aims to support existing hospitality industry business flows, which frequently require their users to juggle various systems to achieve their aims, but without creating a single monolithic system.

2.1. Addressing the challenges

The integration of the domain-specific SaaS solutions in the proposed platform is performed via their APIs. No attempt is made to use techniques such as screen scraping, due to their brittleness. That said, the former solutions are not homogenous, which led to the decision to use drivers to mediate the communication between the core of the platform and the SaaS solutions, much in the same way an operating system talks to printers from various manufacturers.

The SaaS solutions offered in the marketplace are organized in groups – the tool categories in Table 1 – performing (roughly) similar functions. The platform is designed to operate with, at most, one service of each group, except in migration, during which the old and new SaaS services of the same type are active simultaneously.

The set of operations available on each SaaS will vary, not only between groups but among tools in the same group, according to the options of each vendor. To support these differences while maintaining common functionality by the platform requires either a minimum common-denominator approach (foregoing operations that aren't supported by all tools within a group) or allowing divergence among services and announcing the reduced functionality in case the support is absent from the subscribed SaaS service. The choice between which approach to follow is platform dependent, and both may be applied in actual usage according to service characteristics and platform development policy.

The above-mentioned variability calls for a highly dynamic method of generating the user interface, which is made even more complex since each individual hospitality institution may choose its own set of SaaS services to subscribe, and, therefore, the set of operations available to different institutions may vary. That said, training simplification occurs by design since the functions provided by the integrated SaaS solutions are made available to the users via a homogenous interface.

To prevent lock-in, data migration among SaaS providers is a key concern. In fact, the premise of being able to change providers is seriously impaired if this is prevented because of impossible, incomplete, or complex data migration among the services. This problem of migrating data is, however, very hard to solve in general, and although some mechanisms are either proposed or in use for particular domains, like cloud storage [7], no generic solution is so far present in the market. Our proposed architecture creates a migration mechanism based on a common business data repository and migration-specific drivers and tools, to enable the migration of as many data as possible. Even if not completely supported in the destination SaaS, some data can be kept in the platform repository.

Another key concern regards independence from particular components, namely from the property management system – PMS [16] – which in the hospitality industry takes the role usually conceded in other industries to ERP systems [19]. This independence enables the platform to support use-cases and workflows that aren't backed by existing PMS and allows integrating tools not supported by them.

Cloud-based systems can usually support sharing and multi-tenancy, leading to economies of scale and better resource utilization [1]. The literature discusses different approaches of sharing leading up to multi-tenancy: at the data center layer, at the infrastructure layer, or application layer sharing - multi-tenancy [12]. While the proposed architecture could also support a (less common) 4th layer of sharing (where an external application subscription is itself subject to sharing among different users) the actual usage patterns will define whether this approach is suitable (business concerns could e.g. limit this kind of sharing only for subscriptions within the same organization) or if just application layer multi-tenancy is to be supported.

2.2. Users and use cases

The proposed platform will need to support a set of use cases to fulfil its task. There is a need to support domain related operation interfaces for several groups of actors, as depicted in Fig. 1 and described below.

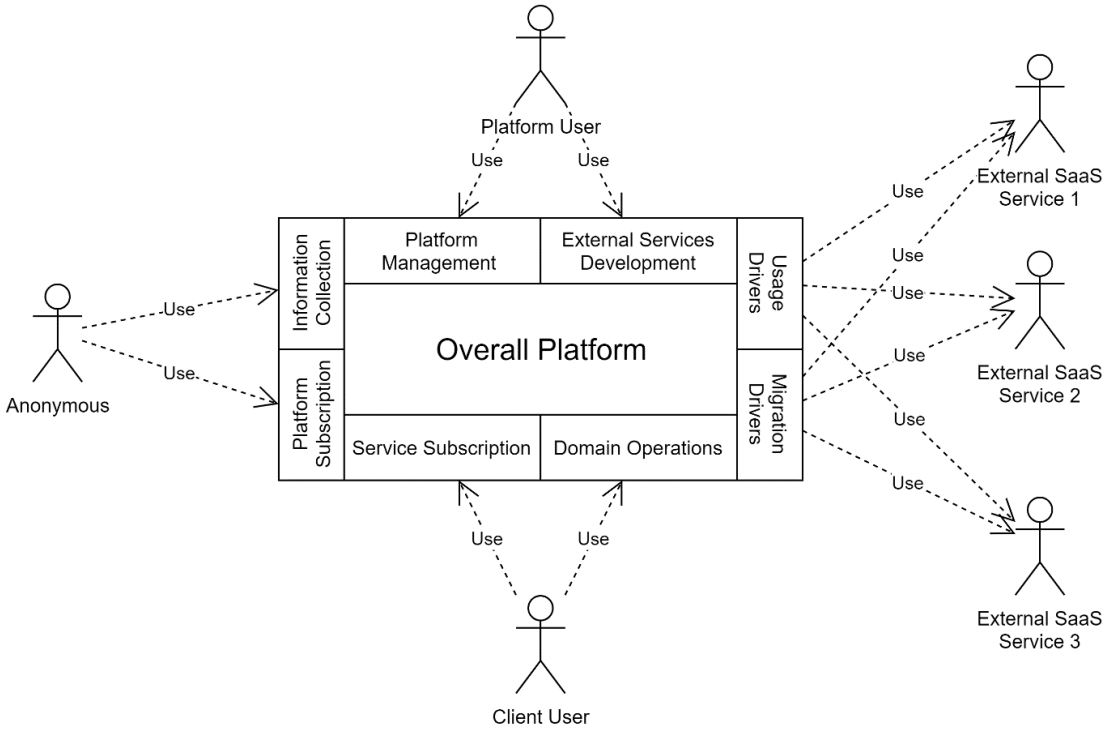

Fig. 1. Actors interaction with the system

- **Client users**: Group of users belonging to a company in the hospitality sector who intend to use the platform to access/manage SaaS services. These can be divided into sub-roles:
    - **Client manager**: an authenticated user who is responsible for administering customer data, for example, managing customer service subscriptions;
    - **Client employee**: an authenticated user who collaborates with a company in the hospitality sector and performs operations (for example, customer check-in) using the services previously subscribed by the client manager.
- **Platform users**: Group of users associated with the development and maintenance of the platform. These can be divided into sub-roles:
    - **Platform manager**: an authenticated user who has a function to control and manage the whole platform;
    - **Platform employee**: an authenticated user who has functions to support management of the platform and/or development of operations/drivers for existing or new services to be supported;

  - **Partner user**: an authenticated user who collaborates with the platform, e.g. to introduce a new service to the Marketplace and provide integration information (including integration and migration drivers) so it can be used;
- **External SaaS Service**: Application that provides operations to the platform, providing an API to support said operations.

To the previously identified actors, we could also add:

- **Anonymous end-user**: an unauthenticated user who wants to learn about the system, and/or authenticate on the platform; It only has (read) access to publicly available information, up to the point it subscribes to the platform.

Since each group of actors has different needs, the user interface must support different operations at each level.

The listed actors exercise a set of use cases, for which different facets of the user interface must be created. These include performing actual operations, delegated to the integrated SaaS solutions, but also the ancillary processes of service subscription, cancellation, and migration. Platform users use-cases, including driver development and updating and new services integration (possibly by external/partner users), as well the marketplace day-to-day management, will be supported by the platform, but their operation is not further detailed in this work.

2.3. Components

The overall platform is designed as a set of cooperating components, organized into functional groups, talking to each other via REST APIs to achieve the desired goals of providing integrated solution out of diverse SaaS services. To do so, each external SaaS functionality is accessed through an *Integration Broker*, via service-specific *Integration Service Drivers* (see top right of Fig. **2**). The precise functionalities required from a SaaS are directed by an *Orchestrator*, which handles the use, subscription, cancellation, and migration of SaaS solutions. For simple use operations commanded by the *User Interface* (e.g. create an invoice), it starts by checking, in the ***Subscriptions Repository***, which SaaS solution is registered as responsible for that functionality and then triggers the operations and invocations required to complete it via the *Integration Broker* and respective *Integration Service Driver*. The ***Integration Broker*** is responsible for redirecting and distributing messages coming from an *Orchestrator* to the *Integration Service Drivers*, and vice versa. As an intermediate component between them, it functions as an API Communications *gateway*, exposing to the *Orchestrator* the endpoints of *Integration Service Drivers*, a Dynamic API corresponding to the operations of the whole of the SaaS services currently subscribed. It also redirects and distributes messages from the *Orchestrator* to the *Integration Service Drivers*, depending on which external SaaS solution a message is meant to. The ***Integration Service Drivers*** are responsible for enabling the communication with the external SaaS solutions, by translating the protocols and business data formats used in the "normalized" internal endpoints to the communication protocols and business data formats required by the various integrated SaaS.

The SaaS solutions available for integration are described in the ***Services Catalog***, exposed as a Marketplace, accessible via the *User Interface*. This component holds both, commercial (e.g. cost, features, user ratings) and technical information (e.g. API details) required for interfacing with them. Once the hotel manager selects a specific SaaS, the *Orchestrator* triggers the necessary steps to make the platform aware of the new functionalities available, including registering it in the *Subscriptions Repository* and notifying *Accounting* and *Billing.* Cancellation is the converse process. Since both can take some minutes, the *Orchestrator* supports notifications via the *User Interface* on the status of the subscription and cancellation processes.

A particular case is that of switching out a SaaS for a "better" one in the same category, such as one Reputation Manager tool for another (see Table 1). In such an instance, there is an interest in migrating as much data as possible from the previously subscribed software to the new one. The substance of this operation is delegated by the *Orchestrator* to the

Migrator, who coordinates the data migration process between external SaaS solutions, using the *Migration Service Drivers,* the *Subscriptions Repository*, the *Data Business Manager*, and the *Accounting & Billing* components. Since the migration process may take some time and require user input decisions, it may trigger related *User Interface* interactions (via the *Orchestrator* component). ***Migration Service Drivers*** are sub-components akin to *Integration Service Drivers*, but with the sole purpose of downloading and uploading data from the external SaaS solutions. When supported by SaaS APIs, these drivers use bulk downloads and uploads of data; if not they mimic that functionality using systematic transfers of individual data records.

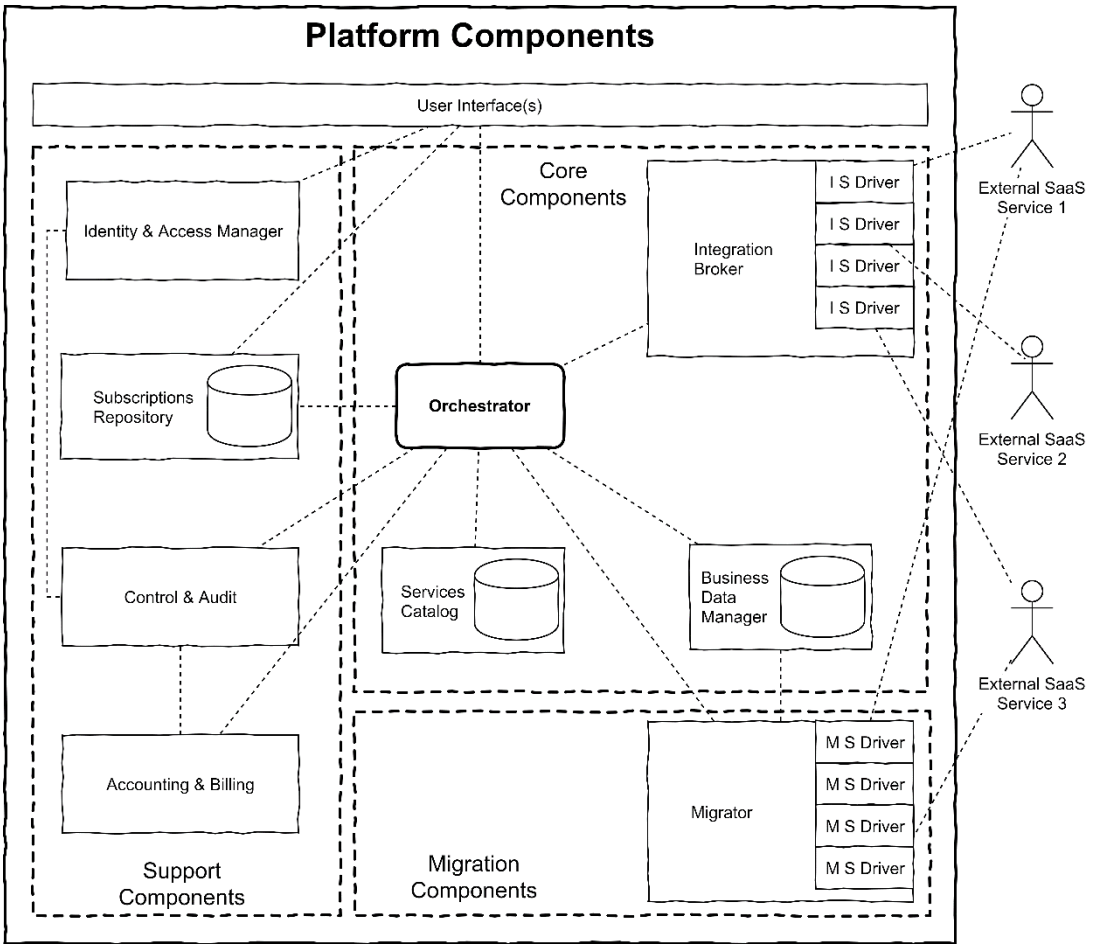

Fig. 2. Platform Components

Since each external SaaS solution may use different representations for domain information, and since some may be useful for cross-service applications, there is a need for a ***Business Data Manager***. This component handles requests for storage and access to business data, supported by a database created according to a "Canonical" business model representation, which should allow for representing all relevant domain information, even if some of it isn't supported by some external SaaS providers. This entails that each canonical data object should be characterized regarding its level of support/requirement for each SaaS service operation present in the system. Although direct access to this component may be permitted for "trusted" architecture components, authentication and authorization are required for externally developed components, like SaaS service drivers.

Besides the previous core components, some additional ones are needed for system operation, namely to support operational security and monetization. ***Identity & Access Manager*** is responsible for control access, with respect to authenticated/authorized and non-repudiable actions. It enables operations of organization management (to support multi-tenancy), management of the organization's roles; user management; control of

operations made available to the various groups of actors, and management of roles permissions and/or user permissions. For monetization, the ***Accounting & Billing*** component is responsible for tracking usage data, to support billing for Clients and Partners. The ***Control & Audit*** component stores and manages logs of the user operations and state of the system components. To this effect, it counts on the collaboration of the remaining components to report relevant data. Not all connections between components are depicted in Fig. **2**, to prevent cluttering.

Finally, the ***User Interface*** component is responsible for generating a browser-based homogenous visual and operational experience for the user, hiding the fact that, "under the hood", several heterogenous SaaS solutions are supporting the functionalities that appear as a seamless integration. The various user roles described in Fig. 1 are also supported by this component. The ***User Interface*** component includes a *User Notification Center*, so other components can perform *push-notifications* to be presented to the user, and also a *Dashboard*, enabling the monitoring of the system use and operation for the different system users.

The overall architecture, although not completely generic, has many degrees of freedom, allowing for supporting various degrees of domain integration, of orchestration complexity and tool endpoints availability. Any further evolution in any of these dimensions can be mostly incremental, without the need to change other components than those directly affected. To do so, development of the canonical model, which is itself created with the ability for change, should follow incremental rather than radical evolution.

3. Tests and validation

The proposed architecture has been implemented and tested for feasibility of the concept and in a real-world pilot.

After the overall architecture was defined, each component (or sub-component) was developed using test-driven development [10]. Test-driven development is an iterative software development technique in which each iteration based on the development of tests (usually on the unit or service testing level). This method proceeds in each cycle by creating simple tests, to verify required functionality, updating the code to provide the required functionality and (optionally) refactoring the code so that redundancies are removed. Iteration is performed by updating each passed test to increase the desired functionality and performing the cycle, at each time without loss of functionality (guaranteed by the previously defined tests).

The different components were implemented in the Rails framework [18] using the Ruby Programming Language, and the tests were described in a custom DSL language supported by RSpec [6], a Behaviour Driven Development gem package, therefore ensuring that the test-driven approach was followed.

The implementation was performed for all the base components described and also for drivers for a set of external services from the categories in Table 1. While integration service drivers were developed for all the external services supported, just a subset of them had migration service drivers developed, due to lack of development resources. The actual number of tests developed depended on the component complexity (e.g. while the Services Catalog required under 150 tests, other components like the Orchestrator and the Migrator required each about 400 tests [2]). The integration of the components was also subject to integration tests, while migration testing required developing a data migration policy and the invocation of several external services to validate the full migration.

To fully validate the approach, the prototype underwent a pilot test in an actual Hospitality accommodation service.

4. Conclusions, Limitations, and Future Work

We described an architecture for an "off-the-shelf" SaaS platform for the hospitality sector capable of integrating several other commercial domain-specific SaaS solutions (e.g. CRM, channel management, invoicing, reputation manage) that the hoteliers can select

from a marketplace, and presenting the result to the user via a homogenous interface that hides the underlying complexity, reducing the learning curve and training costs. Further, the SaaS solutions in use can be replaced at any time and their data migrated to the new ones, thus avoiding vendor lock-in.

The architecture is supported on a "canonical" data model, capable of dealing with the heterogeneity of the SaaS solutions from diverse providers, and on a set of components that interact via REST APIs. Key among these are *Integration Service Drivers*, responsible for interacting with the diversified SaaS solutions, akin to the way an operating system interacts with diversified printers. The modularity of this approach means that new drivers can be created for new SaaS solutions that emerge in the future.

Although instantiated for the hospitality sector, the architecture is generic, so that, given the proper drivers, it can be used to integrate SaaS offerings in other domains of application.

Preliminary tests have proven the feasibility of the concept, and a real-world pilot has also been run. Presently, drivers for various SaaS solutions and some ancillary components are being developed for deployment in a more complex real-world setting.

While the proposed architecture presents a feasible solution to the concerns of the hospitality sector, some limitations may prevent it from achieving the desired outcomes. Chief among them is the dependence of the architecture on the services published and documented APIs. Since many SaaS providers favour interactive usage over remote automation, not all use cases supported by external tools may be available via API. On a previous study, we analyzed 120 SaaS tools for the hospitality industry (on the 15 tool categories depicted in Table 1) and only found 70 of those with API and of those only 39 had API documentation available [15].

Acknowledgements

The work was done under the research project 2017/17692, co-financed by the European Commission framework of structural funds for the period 2014-2020 (Portugal 2020/CENTRO2020). This work has also been partially supported by national funds through the FCT Foundation for Science and Technology, I.P., under project grant UID/Multi/00308/2019 and within the scope of the project CISUC - UID/CEC/00326/2019.

References

1. Adewojo, A.A., Bass, J.M.: Evaluating the Effect of Multi-Tenancy Patterns in Containerized Cloud-Hosted Content Management System. In: 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP). pp. 278–282. IEEE (2018)
2. Amaro, R.J.M.: Concepção e desenvolvimento de uma plataforma de gestão de serviços SaaS para o sector do alojamento - integração e migração de serviços cloud. MSc Dissertation, University of Coimbra (2018)
3. Becerra, M., Santaló, J., Silva, R.: Being better vs. being different: Differentiation, competition, and pricing strategies in the Spanish hotel industry. Tour. Manag. 34 71–79 (2013)
4. Bilgihan, A., Bujisic, M.: The effect of website features in online relationship marketing: A case of online hotel booking. Electron. Commer. Res. Appl. 14 (4), 222–232 (2015)
5. Bulchand-Gidumal, J., Melián-González, S.: Information Technologies (IT) in hotels: A full catalogue. (2015)
6. Chelimsky, D., Astels, D.: The RSpec book : behaviour-driven development with RSpec, Cucumber, and Friends. Pragmatic Programmers Workbench (2010)
7. Duan, Z., Cao, Y., Song, M.: A construction method and data migration strategy for hybrid cloud storage. In: 2015 18th International Conference on Computer and Information Technology (ICCIT). pp. 473–478. IEEE (2015)
8. Ebert, N., Weber, K., Koruna, S.: Integration Platform as a Service. Bus. Inf. Syst. Eng.

59 (5), 375–379 (2017)
9. Garofalo, J.: Why SaaS is Broken (and how we're going to fix it), Blitzen Blog, https://blitzen.com/blog/why-saas-is-broken/, Accessed: November 24, 2018, (2014)
10. Janzen, D., Saiedian, H.: Test-driven development concepts, taxonomy, and future direction. Computer (Long. Beach. Calif). 38 (9), 43–50 (2005)
11. Kim, M., Lee, S.K., Roehl, W.S.: Competitive price interactions and strategic responses in the lodging market. Tour. Manag. 68 210–219 (2018)
12. Krebs, R., Momm, C., Kounev, S.: ARCHITECTURAL CONCERNS IN MULTI-TENANT SaaS APPLICATIONS. In: Proceedings of the 2nd International Conference on Cloud Computing and Services Science. pp. 426–431. SciTePress - Science and and Technology Publications (2012)
13. Law, R., Leung, R., Lo, A., Leung, D., Fong, L.H.N.: Distribution channel in hospitality and tourism. Int. J. Contemp. Hosp. Manag. 27 (3), 431–452 (2015)
14. Merkel, D., Santas, F., Heberle, A., Ploom, T.: Cloud Integration Patterns. In: Service Oriented and Cloud Computing. ESOCC 2015. Lecture Notes in Computer Science, vol 9306. pp. 199–213. Springer, Cham (2015)
15. Miranda, J.C.G.: Estudo e desenvolvimento de uma plataforma de gestão de serviços SaaS para o sector do alojamento - subscrição e cancelamento de serviços. MSc Dissertation, University of Coimbra (2017)
16. Newhouse, W., Ekstrom, M., Finke, J., Weeks, S.: Securing Property Management Systems - Cybersecurity for the Hospitality Sector. (2017)
17. O'Connor, P., Merten, R., Eisenbeis, F., Sileo, L.: Independent Lodging Market: Marketing, Distribution and Technology Strategies for Non-Branded Properties. (2015)
18. Ruby, S., Copeland, D.B., Thomas, D.: Agile web development with Rails 5.1. Pragmatic Programmers Bookshelf (2017)
19. Serdeira Azevedo, P., Romão, M., Rebelo, E.: Success factors for using ERP (Enterprise Resource Planning) systems to improve competitiveness in the hospitality industry. Tour. Manag. Stud. 10 165–168 (2014)
20. Sun, W., Zhang, K., Chen, S.-K., Zhang, X., Liang, H.: Software as a Service: An Integration Perspective. In: Service-Oriented Computing – ICSOC 2007. pp. 558–569. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
21. Thakran, K., Verma, R.: The Emergence of Hybrid Online Distribution Channels in Travel, Tourism and Hospitality. Cornell Hosp. Q. 54 (3), 240–247 (2013)
22. The Economist: Sun Sea and Surfing: The market for booking travel online is rapidly consolidating, https://www.economist.com/business/2014/06/21/sun-sea-and-surfing, (2014). Accessed August 20, 2019
23. Tsai, W.-T., Sun, X., Balasooriya, J.: Service-Oriented Cloud Computing Architecture. In: 2010 Seventh International Conference on Information Technology: New Generations. pp. 684–689. IEEE (2010)
24. Zhang, L.-J., Zhou, Q.: CCOA: Cloud Computing Open Architecture. In: 2009 IEEE International Conference on Web Services. pp. 607–616. IEEE (2009)