On the Future of Solution Composition in Software Ecosystems

Zherui Yang[1], Slinger Jansen[1], Xuesong Gao[2], and Dong Zhang[2]

[1]Utrecht University, Utrecht, The Netherlands
y.z_ryan@hotmail.com, slinger.Jansen@uu.nl
[2]Huawei Technologies CO., LTD , Beijing, China
{james.gaoxuesong, zhang.dong}@huawei.com

Abstract. The trend of application stores is currently at a peak. However, the lack of dynamic composition for complex solutions is the largest downside of the app store model, since solutions are increasingly created as compositions of multiple solutions, APIs, and applications. Therefore, in this vision paper, a superior model, *solution composers*, is proposed to the app store model. A conceptual framework is established to illustrate the inner workings of solution composers in software ecosystems. In order to outline that solution composers are significant for the future of software development, several industry cases are presented and compared to support this concept, further indicating that a standard for solution composition should be considered. In addition, the vision is evaluated through expert reviews at several leading platform providers and challenges for practice and implementation are identified.

Keywords: Software ecosystems, AppStores, solution composers

1 Introduction

Software ecosystems are complex networks of organizations, that collaboratively serve a market [12]. Long value chains are formed in these ecosystems. The actors in these networks are Software Producing Organizations (SPOs), such as open source consortia and Independent Software Vendors (ISVs), and end-users, such as software consumers of mobile apps, or employees at large companies that require advanced business applications.

Whereas in the past end-users would accept pre-configured monolithic solution bundles in the form of, for example, large Enterprise Resource Planning applications or complicated mobile apps, increasingly there is a demand for flexible compositions of solutions from end-users that can be changed rapidly and dynamically. In many cases, even non-technical end-users want to compose such solutions. Therefore, in order to meet this demand, in this paper, the concept of a *solution composer* is proposed and a conceptual framework is presented.

Research and studies from previous period have gained promising and valuable results, but have mostly focused on specific aspects of solution composition or service integration [8, 17], which are fragmented and lack of a holistic view of the concept or the problem [14, 4, 6]. Therefore, a more holistic and consolidated framework is needed that attributes to the fundamental understanding of solution composers in terms of concepts, industry practices, innovation and development direction. This framework is useful and necessary to enhance the efficiency and effectiveness of solution composer establishment.

There is an emerging need for customization and composition of solutions. Firstly, there is an increase in interfaces and devices that provide access to valuable features. Secondly, platforms are increasingly the gateway to large collections of these features, such as Internet of Things (IoTs) platforms [2]. Pre-composed and configured feature bundles, such as apps in the current AppSotres, are rapidly losing their value, as these monoliths cannot easily be adapted to create new solution compositions. One illustration of this development is the Android Instant Apps platform [19], where apps can be downloaded and activated based on a set of predefined events, without having to use the Play Store. This paper focuses on the following research questions:

1. *Is there a need for solution composers?*
2. *Is the proposed solution composer framework valid?*
3. *What will the industry implications be of the introduction of solution composers?*

This paper continues with the research method, i.e., a number of industry case studies and evaluation interviews. In Section 3, a critical discussion is given on the current software industry. In section 4, the solution composer concept is introduced and the proposed framework is described in detail. Moreover, the implementations of the framework in industry are discussed. Experts were invited to evaluate the framework in forms of interviews in section 5. Final discussion is in section 6 and we provided conclusions and an outlook in section 7.

2 Research Method

This research can be seen as a light mixed study based on case study and interviews, using design science [24]. First, several industry cases were studied and compared to establish a conceptual framework for solution composer. Second, expert interviews were conducted to evaluate both the theoretical and practical aspects of solution composers.

The design research paradigm focuses on creating and evaluating innovative IT artifacts that enable organizations to address important information-related tasks [24]. This paper aims to determine the nature of solution composers. The research method is deemed appropriate when there is little evidence about a phenomenon and the researcher seeks answers to research questions [20]. The authors looked for experts who have a great deal to share about service composition and software ecosystems.

The cases were selected as representative service composition tools from the industry. And the interviews were conducted in a semi-structured way. This method has been used to explore the framework, allowing new ideas to be brought up during the interview as a result of interviewees' answers [26]. All the authors contributed to the interview questions and the interviews were conducted by one of the authors. All the evidence including audio tape, transcripts and notes from the interview have been kept and these information was used to compose this report [26].

3 The Limits of AppStores

Jansen and Bloemendal [11] define an AppStore as an online curated marketplace that allows developers to sell and distribute their products to actors within one or more multi-sided software platform ecosystems. AppStores have striking advantages for the software business. They have changed the application industry significantly [11], introducing new business and deployment models to partners, and offering end-users the maximum freedom of choosing applications. Also they provide app developers with a wealth of opportunities to approach new niche markets with domain specific applications [10].

Despite the advantages AppStores have, the AppStore model is increasingly inflexible. Software applications in AppStores are monolithic collections of features that aim to solve problems for end users, whether it is to book a train ticket or to provide fun through a game. The downside of functionally composing such features into a collection is that end-users do not have the flexibility to break down those features and compose them into novel solutions. End-users are burdened with the job of carrying over content from one application to the next manually and are tired of the lack of dynamic composition for complex solutions. It appears AppStores have reached their peak. It is time for the "mini-monoliths" in the AppStore to be decomposed and offered as separate features to solution composers.

4 Solution Composers

The state of the practice introduces four types of software composition framework: developer platforms, API composers, SOA composers and orchestrators, and end-user service composers. After learning from and comparing with these frameworks, a conceptual framework of solution composers is proposed, highlighting necessary elements in any successful solution composer. Subsequently, several cases are presented and analyzed as the implementations of the proposed framework in industry.

4.1 State of the Practice

The following four types of software composition frameworks are identified that contain elements of solution composers.

Developer Platforms Developer platforms [23] are platforms that enable developers to create complex solutions on top of existing platforms, typically with a rich set of developer tools to enable quick adoption of the platform. In the context of solution composers, they enable developers to integrate services from the ecosystem into their solutions, however, the facilitation of such composition is typically through regular programming.

Example: Force.com is Salesforce's developer platform, that enables developers to create domain specific solutions on top of Salesforce.com. In Force.com we find that developers integrate services from the platform and other APIs manually. It has an AppStore for end customers, but this does not enable automatic or supported composition of services and developers simply address the APIs of the apps.

API Composers API composers help to manage APIs to fast, affordably and scalably communicate between all the values that have already been built down and the new projects that are building for today, analyzing API data and handle the IoT. In the context of solution composer, they allow separate functions to communicate with each other but they do not provide internal or external services for solutions composition.

Example: Apigee is an API composer, which allows end-users to secure, manage, analyze and connect all APIs. Although it enables end-users to manipulate APIs and provides API management to store all compositions, there is no service index in Apigee for end-users to choose available services from.

SOA Composers and Orchestrators At a very high level, a crucial aspect of Service Oriented Architecture (SOA) is service orchestration. Enterprise systems and integration projects designed according to SOA principles depend on successful service orchestration. In the context of solution composer, SOA composer is a web-based application that helps to build new applications with granular and reusable software components, but it lacks the flexibility and agility to dynamically compose and integrate smaller units of services.

Example: Oracle SOA Composer allows users to work with Oracle Business Rules dictionaries and tasks for deployed applications. Moreover, Oracle SOA Composer provides a platform with enhanced service orchestration capabilities to ease the integration challenges, but still not flexible enough for dynamic complex solution composition.

End-user Service Composers End-user service composers are platforms or web services that aggregates many other web services or applications into one place and can then perform actions given a certain set of criteria. End-users may therefore create customized criteria and perform the action as desired. Because of the customization character, end-user service composers might be the most similar one to solution composers, nevertheless, they are too user-friendly for developers to integrate and compose more complex solutions.

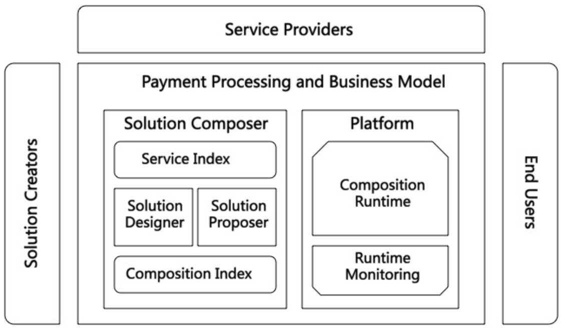

Fig. 1. The Solution Composer Framework

Example: IFTTT.com is a free web-based service that allows users to create chains of simple conditional statements, called "recipes", which are triggered based on changes to other web services such as Gmail, Facebook and Instagram. It has the necessary elements for solution composer, but it is way too immature for higher level of solution composition to be performed on.

4.2 The Solution Composer Framework

By learning from and combining with the software composition frameworks, a conceptual framework of a solution composer platform is proposed in Figure 1, in order to create a better understanding of how solution composer performs.

From the top, left and right sides of the figure, the different actors within the framework are modeled, respectively the service providers, the solution creators that might be end-users or consultants in the companies, and the end-users.

The communication between actors and the core of solution composer in Figure 1 is payment processing and business model. When solution composer is carried into practice, the business model will therefore connect the theoretical framework to the industry.

In the center of this framework, solution composer represents how it works in detail with features as service index, solution designer, solution proposer and composition index. Service index stores all available services and presents to end-users in a certain form such as *Library* or *Service Catalog*. End-users can select a range of services or applications for the need of integrating and composing services according to their specific need. Solution designer is where end-users can get their selected services and applications designed in some pattern which is suitable for the final solution while Solution proposer is an entity providing solution propositions. The composition index is created to store compositions and benchmarks of solutions. The compositions are required to provide a list of solutions that have been created in the past and can now be reused. The solution benchmarks are especially useful for larger solution compositions where some knowledge is required about how the solution is going to perform in the future. In the field of Software Defined Networking, for instance, bringing together different parts of a solution is challenging, as little may be known about the performance of (a combination of) services. Having a set of benchmarks can remedy this situation to bring some predictability in the process of solution composition.

We separate solution composer and platform because solution composition is the phase where service composition is designed and proposed while composition run-time is the phase where the service composition is installed in the run-time environment and executed. When the service composition platform enables the selection process of individual services at deployment time, usually the composition from composition index can be re-configured. Moreover, the run-time monitoring will monitor and analyze how the service composition is executed and get as much performance evaluation as possible.

4.3 Implementations of the Solution Composer Framework in Industry

Nowadays, business and technology can turn an idea into a potential product, a new service or a better experience in a blink. With such fast growing, customers find it more difficult to get satisfied outcomes about market placement and development strategy [5], along with the related risk issues [13]. Also, as different SPOs attempt to produce complex combination of software systems and hardware [9], there is a need for manual-automatic-combined solution composition.

Different from applications from AppStores, where apps work as individually separate collections of features, the process of the solution composition and integration will require communications and interactions between features. Enterprise application integration has traditionally relied on software-based middleware, such as Service Oriented Architecture (SOA) middleware solutions [16]. Disadvantages of SOA middleware, such as the lack of standards, high cost and the inflexibility [3], make solution composers more appealing. Based on the ideals of middleware, solution composers provide an open, standards-based approach to integration. Unlike its predecessor, the Application Programming Interfaces (APIs) used in solution composers is not a piece of software. Instead, it is a fully functioning integration point. An API is much more flexible and agile than any existing set of routines, protocols, and tools for the purpose of connection [7].

As the possibilities to create service compositions become more complex, more technically oriented resources are required to create new solutions. Furthermore, as third parties will probably also provide basic APIs that are compatible with the platform, advanced mechanisms are required such as service indexes and semi-automatic service composition. Both of these lead to intricate value chains and software ecosystems with many participants in them.

For solution composers to present solutions to customers, four different ways are observed as manually through code (Manually), through a composition studio (Composition Studio), through a composition proposer (Composition Proposer), and hybrid (Hybrid) combinations of these three.

These four ways may be in different combinations to support end-users and composers optimally. In the first way, developers compose solutions with code,

Table 1. Implementations of the service providers, the Solution Composer, and End-users in Industry. ISV stands for Independent Software Vendor, IoT stands for Internet of Things.

Company	Party		
	Online Service Providers	Solution Composers	End Users
Android	API Providers	APP and API provider	End users
APIGEE	API provider	Developers	Everyone
Azure	Cloud service provider	Cloud solution provider/developers	Companies
HP SDN	ISVs/HP open source	HP Consultants	Companies
IFTTT.com	IoT/API providers	End-users	End-users
Pipemonk	ISVs	Developers/End-users	Companies
Salesforce.com	ISVs	Consultants	Companies
We-wired web	IoT/Web apps	End-users	End-users
X-Formation Connect	IoT/API providers	developers	End-users
Zapier	IoT/API providers	End-users	End-users

for instance by programming against APIs from third parties and combining them to create innovative solutions. In the second way, a complete composition studio is offered that enables developers, technical consultants, and even end-users to create new service configurations to create the best fitting solution. The third way does not actually lead to a configuration, but proposes a solution beforehand, which for instance enables benchmarking and comparison of different service configurations. The fourth way is a combination of the first two, where simple configurations can be created, but more advanced solutions still have to be coded traditionally.

In Table 1, several tools are introduced for comparative analysis. The tools evaluated were selected using a Google search for service composition tools. The tools needed to satisfy the following criteria: (1) Enable the composition of services to create a solution, (2) have a service index for the creation of solutions, (3) be exemplary in the industry, and (4) be commercially available. We grouped the API aggregation platforms, as there are many new entrants to the market.

In Table 2, cases are further compared based on the four different features in solution composers: service indexes, solution designers, solution proposers, and composition indexes. These four features define the character of a solution composer. Service and composition indexes indicate where users can find all services and composition available on the platform. The main features of solution composers are solution designer and proposer.

As for an example, IFTTT allows end-users to create, integrate and combine services into solutions and store these chains of simple conditional statements as recipes, or as we call it here composition index. Thus, IFTTT consists of service index (channels), solution designer (interface), solution proposer (end-users themselves) and composition index (recipes), which makes it actually one of the first to fully implement a solution composer. Also, Zapier shares the same construction with full implementation of a solution composer. However, with a more extensive service index and more flexible solution design patterns, Zapier, to an extent, is even one step closer to the ideal implementation of a solution composer.

Table 2. Solution composer features observed in industry

Case	Party			
	Service Index	Solution Designer	Solution Proposer	Composition Index
Mashup [15]	UDDI Service Catalog	Mashup environment	None	None
FeatureHouse [1]	Tree index	None	None	None
Android	APP Store	None	None	None
APIGEE	None	Customer's own IDE	None	API management
Azure	Runbook gallery	Microsoft Powershell	End-users/None	None
HP SDN	SDN APP Store	None	None	None
IFTTT.com	IFTTT.com Channels	IFTTT interface	End-users/None	Recipes
Pipemonk	Shopify AppStore /Amazon Sellers	Pipemonk interface	End-users/None	QuickBooks
Salesforce.com	AppExchange	App developer IDE	None	None
We-wired web	Service Catalog	Visual wiring diagrams	End-users/None	None
X-Formation Connect	Application drop-list	Connect interface	End-users/None	None
Zapier	Library	Zapier interface	End-users/None	Zaps

For some tools, such as Pipemonk, We-Wired Web and X-formation Connect are more or less like IFTTT or Zapier. They share the idea of service integration automation and solution composition. What is different is that these tools do not have a handy composition index for end-users.

For the rest of the cases, they all miss some essential elements, but each of them contains significant part(s) of a solution composer. APIGEE is a API management platform, focusing more on the designer and composition index part. Microsoft Azure is a growing collection of integrated cloud services. It provides cloud services for the need of end-users but it does not have a composition index. HP SDN allows end-users to select from a range of SDN Applications that allow to program network to align with business needs. But HP SDN only has an AppStore for the selection, and arranging consultants to help with all the solution. There is no reference solution or relevant database for composed solutions. Salesforce.com is a developer platform. It only provides service index and solution designer.

After an extensive literature study, a couple of scientific frameworks were selected by conducting snowballing procedure [25]. Snowballing refers to using the reference list of a paper or the citations to the paper to identify additional papers. The FeatureHouse framework [1] provides a method for software composition using superimposition. The framework, however, only concerns the composition of systems from different languages and does not provide tooling for suggestion of fitting solutions, nor does it provide an index of composed solutions. Another framework that was added is the Mashup framework [15], which is a mechanism for enabling end-users to create mashups from a UDDI registry of services, using drag and drop tools.

5 Evaluation

Based on learning and results from the industry cases, an interview question protocol was drawn. In the second phase of this study, semi-structured interviews

Table 3. The background of interviewees

Background and occupations	Interviewee
CEO of the company	A
CTO and scientist innovator	B
Product manager ecosystem	C
Cooperation manager	D
Senior developer	E

were conducted. During interviews, experts were asked to provide insight on the concept of solution composers and to evaluate the proposed framework. The interviews were recorded, after which all the records were transcribed.
Interviewees' Backgrounds - Interviewee A owns a company and is the CEO of the company for three years. The projects he has been working on based mostly on the idea of service composition. Interviewee B works as scientist innovator in a project based national company for almost 4 years and he may join a new project regarding service composition. And interviewee C works as a product manager ecosystem in a software company for nearly two years. While interviewee D works in a mobile company and is handling a project closely related to solution composers. And interviewee E is an developer in a e-commerce company with experience in the filed of web service. All backgrounds are indicated in Table 3.
Interview Analysis - Due to the wide range selection of questions, not all of the interviewees were able to answer 100% of the questions listed. However, the authors were able to combine and compare the information among all the interviews and draw conclusions about the validity of the proposed framework and the two-sides of solution composers. In this section, a thorough discussion and analysis is provided.

The need of solution composers During interviews, interviewees indicated that there indeed is a need for solution composer in the software industry that will fulfill the need of end-users. Solution composer has further affected the software ecosystem by providing standards and creating a new market. Currently, some significant big shots in the software industry have started developing similar services, such as Android Instant Apps platform.

Solution composers will offer standards and protocols to support the communication among component services. However, current service composition environments barely have productivity support tools which is similar to what modern Integrated Development Environments (IDEs) provided, such as code searching or debugging [14]. Solution composer could therefore benefit from environments with productivity techniques, for example, services index discovery and services integration. As interviewee B also confirmed in the interview, "*there is a lack of standards*". Thus, solution composer that provides standards is needed for services integration and composition.

Moreover, there is a market need for solution composer. With regards to the effect of solution composer, interviewee A, when asked about the impact on software ecosystem, replied:" *Its going to create a whole new way of apps*

Table 4. The need, advantage, challenge and validity of solution composers: Quotes from the Evaluation Interviews

On the need for solution composers	**Inter-viewee**
”Change the industry”; ”Offer completely new market place”.	A
”It will create a profitable market for whoever is in part of this revolution”.	B
”Obviously, there is a need for end-users”; ”its really getting there already”.	C
”It can reach out to many fields and can be used in many ways. The ICT area will be affected”.	D
”Bring in new concept”; ”a big innovation”; ”will actually build healthier software ecosystems”.	E

On the advantage of solution composers	**Inter-viewee**
”Dont think of Apps anymore”; ”make it easier for the users to get functions they need”.	A
”Making it easier for the end-users”.	B
”Allow end-users to customize”; ”can make business around it”.	C
”First, it includes a service consultancy. Secondly, it provides end-users with an experiential environment”; ”attainability”.	D
”The standardization and the attempt of customization”; ”divide two phases of the design and the run-time”.	E

On the challenges of solution composers	**Inter-viewee**
”Privacy is maybe still an issue. ’Do you want it?’”.	A
”Practical problems”; ”need well-defined APIs and standardized”.	B
”Standardization”; ”even without standardization, you will need to build an ecosystem”; ”90% of what you need is person (manual work)”.	C
”The technique support”; ”how to simulate the environment”.	D
”How you can persuade people to use”.	E

On the validity of the framework	**Inter-viewee**
”Could work for the business”; ”(will need to) customize their daily operations and the new technical infrastructure”.	A
”Architecture for a framework that has not been implemented”; ”A lot of manual work to get everything to work on their platform”.	B
”It sounds technical and detailed”; ”can already be valuable now”.	C
”This is a good idea”; ”will add value”.	D
”Quite clever”; ”will make things easier and smoother”.	E

(services) and not only apps”. He also suggested that some of the big companies with their own ecosystem have got hands on this field already, ” *Google, for example, is already creating this alternative AppStore*” ,” *its just about the first party who gets in the market as fast as possible*”. In the meanwhile, interviewee B agreed on this point.

Solution composer is more like a trend, rather than a tool needed to be developed or introduced to the industry. We foresee that the trend of composing software from small units of functionality will continue.

The advantages of solution composers Solution composer focuses on single applications no more. To some extent, solution composer represents a higher level of functionality and service composition. The idea of service composition indicates the future of software industry, which is to meet the need of end-users. This is also the most important aspect of software development. Thus, thinking of the end-users is the main advantage of solution composer, as interviewees all agreed.

Moreover, regarding the aspect of solution composer operation, interviewee D pointed out that solution composer can provide end-users with an experiential environment, allowing end-users to sense the final product they are going to purchase. In this way, a solution is proposed in advance before it is carried out into practice. Therefore, benchmarking and comparison of different service configuration are enabled. "*If the end-users could have the access to a trial product with consultant and specialist's advice, they may feel they've reached closely to their goal*". This approach to present composed solution can soon meet end-users requirement and satisfy end-users expectation. For end-users, the actual outcome helps to make their finally decision.

The challenge of solution composers Nevertheless, challenges of the solution composer exist. The main concern addressed from interviewees is whether solution composer can successfully attract customers in the current market. As a matter of fact, with such a strong idea colliding with the AppStore nowadays, it is reasonable to have concerns over the result of solution composer reshaping the industry.

The interviewees reflected that standardization, which was also implied as predefined manual work, would cost a lot of time and labor. In addition, interviewee C brought up that implementing solution composer within an ecosystem is also something needed to be concerned. Nevertheless, solution composers need to be built on top of an existing ecosystem.

The validity of the proposed framework In the framework, we defined solution composer and platform as separated phases because they have different focuses. During the interview, Interviewee E agreed on this separation.

Furthermore, the proposed framework provides a standard for service composition that the current industry is lack of. "*I would say there is a lack of standards*", said interviewee B, "*While some are very similar services but they just have different APIs and there are some much work to implement that specific API*". The problem interviewee B brought up is what the solution composer is about to fix. We proposed service index in order to form a standardized interface for end-users, gathering all the services together rather than a whole bunch of scattered APIs. In addition, the composition index brings convenience as well. It allows end-users to easily look up composed solutions. interviewee E said that the composition index could be very interesting.

In the framework, besides the technical side, we also included payment processing and business model in order to make it valid for business that will be the work in the future.

Moreover, interviewee B suggested how we should make the framework more valid or more advanced. "*If you have well-defined services*", he gave suggestions on how the solution composer could support semi-automatic composition, "*if you have some precondition and some post condition, output and input then you could create an engine to do this*".

However, interviewee B also addressed his concerns as the payment processing and business model being in the center of the framework. He said:"*If you are saying business models and payment processing, then you are talking about something in companies or back-ends.*" In the meanwhile, interviewee A also had the same concerns about whether there is an appropriate ecosystem or a business background to support such framework, to bring it into practice and to make it profitable in the market.

6 Discussion

Implications for the Industry. In order to validate the framework for practical implementations in software industry, protocols and standards are needed for services to communicate. Within an environment for solution composer, different requirements in terms of component models are needed [14]. Since every service works differently, it will be quite a burden if there is no protocol for communications among services and consequently, it will increase the difficulty in the solution composition process. For a simple example, booking tickets for flight, every website is different. If there is standardized interface for travel information, it will make things much easier. Nowadays, some very similar services have different APIs and there will be abundant of manual work to implement specific APIs if there is no unified protocol.

Moreover, software services should be simplified to only focus on core features. With clear and distinguishable core features and without unnecessary communications among redundant functions, it is easier for services to follow the standardized protocol and for solution composer to perform concisely. Interviewees B also expressed the urge for the industry to simplify software services.

In the meanwhile, automation is also needed. Nowadays, enterprises are increasingly looking for new chances to cooperate with other enterprises by offering and performing integrated services and solutions [21]. However, the development pace of solution composer that requires a considerable effort of low-level programming has not kept up with the rapid growth of available opportunities. Besides, the number of services to be integrated and composed may be huge, so even with a standard protocol for component communication, involving significant amount of manually coding work is inadequate considering the scale of solution composition. Therefore, solution composers call for automation.

In addition, when automation is included and manual work is reduced, more time and effort could be devoted into the main useful functional part of solution composer, which is solution composition and service integration. Also with a standardized protocol and simplified core service features, it will therefore make consultancy easier and consequently will enable rapid system integration.

Last but not least, in order to facilitate the development of solution composer, a healthy existing ecosystem is significant, either basing solution composer on an ecosystem, or building an ecosystem around solution composer. Only within an ecosystem, solution composer can be made the best use of, nourishing the health of the ecosystem.

Research Validity. This paper brings up with a new concept of solution composer. In order to further investigate the nature of it, we used exploratory research based on case studies from the industry [22] and interviews from experts [18].

With regards to internal validity, evaluating the framework with interviews was a pragmatic decision, since the implementation and testing of the framework in practice requires years of research. We do plan to implement the framework, however, over the course of the next years at several industry research partners.

Regarding external validity (generalizability), it refers to the extent to which the framework of this study can be generalized in industry. As observed in the interviews, this framework has been evaluated in a variety perspectives and the model can be applied to different parts of the software industry, such as the services business or the Internet of Things business, as platforms like IFTTT.com illustrate. Moreover, to minimize the external validity, we also analyzed practical cases from industry in order to show the general practice of the proposed framework.

In terms of construct validity, the interviews were prepared with an extensive interview protocol, consisting of a structure for the interview, but also definitions for the solution composer and its parts. Using this definition list, interviewees were sure to understand the concepts in the same way as other interviewees. The semi-structured interviews were conducted as part of the case study. An interview protocol was defined with questions including status quo in enterprise AppStore and service composition, the impact of solution composer on software ecosystem, and the future of service composition and software ecosystems. The interviews were recorded, and transcribed. The results were analyzed to extract observations, improvement suggestions, and conclusions.

7 Conclusion

From the industry cases and expert interviews suggested in the previous sections, it is clear that solution composers can be a major game changer in shaping and reshaping the complete software ecosystem. One of the main observations drawn from this study is that there is an undoubted need for the development of solution composers, which provide end-users with more relevant and satisfactory solutions. Furthermore, the proposed framework was considered useful for industry practice, according to the experts from related fields.

This paper functions as an exploratory study into solution composers and as a call for practitioners and researchers to further investigate solution composers in practice. The industry cases illustrate that current SPOs are working towards solution composers. According to the proposed framework, most of the participants of this evolution movement for service composition have not yet developed into maturity. Only a few have made the first baby steps towards solution composers, yet still need abundant guidance and instructions to fully grow into the real ones. Also the cases demonstrate, by performing solution composition, SPOs are actually benefiting and gaining market. In the meanwhile, the scientific frameworks from previous works indicate the implementability of solution composers.

In addition, expert interviews provide insight and evaluation from practical side and help to evaluate the research results. Interviewees expressed that there is a need for solution composers and the main advantages focus on customization for end-users and the new way to discover and connect software in the wider software ecosystem. However, most of the challenges brought up were from technical and practical side, which present directions for future research.

First, both industry cases and interviews suggest that in order to establish a robust solution composer, predefined standard will be needed, for instance, the standard protocol for component services to communicate with each other while composing solutions. Secondly, despite the need from market, solution composers will suffer pressures from other existing big shots in the industry. Whether thrive or not, depends largely on how solution composer will be unveiled by whom. Thirdly, persuading end-users to join this software evolution will encounter obstacles and barriers, because end-users may not realize how eager they need the existence of solution composer. However, we will leave these to future research and studies.

References

1. Sven Apel, Christian Kastner, and Christian Lengauer. Language-independent and automated software composition: The featurehouse experience. *Software Engineering, IEEE Transactions on*, 39(1):63–79, 2013.
2. Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.
3. Afkham Azeez, Srinath Perera, Dimuthu Gamage, Ruwan Linton, Prabath Siriwardana, Dimuthu Leelaratne, Sanjiva Weerawarana, and Paul Fremantle. Multi-tenant soa middleware for cloud computing. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 458–465. IEEE, 2010.
4. Jeppe Brønsted, Klaus Marius Hansen, and Mads Ingstrup. A survey of service composition mechanisms in ubiquitous computing. In *Workshop on Requirements and Solutions for Pervasive Software Infrastructures*, volume 2007, pages 87–92, 2007.
5. Ronni Colville, Patricia Adams, and Debra Curtis. It service dependency mapping tools provide configuration view. *Gartner Research News Analysis. Gartner*, 2005.
6. Mohamad Eid, Atif Alamri, and Abdulmotaleb El Saddik. A reference model for dynamic web service composition systems. *International Journal of Web and Grid Services*, 4(2):149–168, 2008.
7. Lee Garber. The lowly api is ready to step front and center. *Computer*, 46(8):14–17, 2013.
8. John Garofalakis, Yannis Panagis, Evangelos Sakkopoulos, and Athanasios Tsakalidis. Web service discovery mechanisms: Looking for a needle in a haystack. In *International Workshop on Web Engineering*, volume 38, 2004.
9. Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of software engineering*. Prentice Hall PTR, 2002.

10. Sami Hyrynsalmi, Tuomas Mäkilä, Antero Järvi, Arho Suominen, Marko Seppänen, and Timo Knuutila. App store, marketplace, play! an analysis of multi-homing in mobile software ecosystems. *Jansen, Slinger*, pages 59–72, 2012.
11. Slinger Jansen and Ewoud Bloemendal. Defining app stores: The role of curated marketplaces in software ecosystems. In *Software Business. From Physical Products to Software Services and Solutions*, pages 195–206. Springer, 2013.
12. Slinger Jansen, Michael A Cusumano, and Sjaak Brinkkemper. *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry.* Edward Elgar Publishing, 2013.
13. Slinger Jansen and Wilfried Rijsemus. Balancing total cost of ownership and cost of maintenance within a software supply network. In *proceedings of the IEEE International Conference on Software Maintenance (ICSM2006, Industrial track), Philadelphia, PA, USA*, 2006.
14. Angel Lagares Lemos, Florian Daniel, and Boualem Benatallah. Web service composition: A survey of techniques and tools. *ACM Computing Surveys (CSUR)*, 48(3):33, 2015.
15. Xuanzhe Liu, Yi Hui, Wei Sun, and Haiqi Liang. Towards service composition based on mashup. In *Services, 2007 IEEE Congress on*, pages 332–339. IEEE, 2007.
16. Qusay H Mahmoud. Service-oriented architecture (soa) and web services: The road to enterprise application integration (eai). *Retrieved November*, 16:2005, 2005.
17. Anbazhagan Mani and Arun Nagarajan. Understanding quality of service for web services. *IBM developerWorks*, 1, 2002.
18. Nick Midgley, Sally Parkinson, Joshua Holmes, Emily Stapley, Virginia Eatough, and Mary Target. did i bring it on myself? an exploratory study of the beliefs that adolescents referred to mental health services have about the causes of their depression. *European Child & Adolescent Psychiatry*, pages 1–10, 2016.
19. Emil Protalinski. Google unveils android instant apps that launch immediately, no installation required. http://venturebeat.com/2016/05/18/google-unveils-android-instant-apps-that-launch-immediately-no-installation-required/. Accessed: 2016-05-18.
20. Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131–164, 2009.
21. Quan Z Sheng, Boualem Benatallah, Marlon Dumas, and Eileen Oi-Yan Mak. Self-serv: a platform for rapid composition of web services in a peer-to-peer environment. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 1051–1054. VLDB Endowment, 2002.
22. Robert A Stebbins. *Exploratory research in the social sciences*, volume 48. Sage, 2001.
23. Joey van Angeren, Carina Alves, and Slinger Jansen. Can we ask you to collaborate? analyzing app developer relationships in commercial platform ecosystems. *Journal of Systems and Software*, 113:430–445, 2016.
24. R Hevner von Alan, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004.
25. Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, page 38. ACM, 2014.
26. R.K. Yin. *Case study research: Design and methods*, volume 5. Sage Publications, Incorporated, 2008.