Reflections over Communicability in PaaS Environments

Rafael Brandão, Marcio Moreno, Juliana Ferreira, and Renato Cerqueira
IBM Research
Rio de Janeiro, Brazil
{rmello, mmoreno, jjansen, rcerq}@br.ibm.com

***Abstract*—Platform as a Service (PaaS) has become an essential product for large technology companies. It is a way of delivering hardware, software tools and other resources for application development and hosting, as a service. Its users are developers who need to build and deploy new applications. Besides computational power, PaaS environments (PaaSE) offer services, development tools and even complete apps to be put together in web applications. These pieces of software can be developed by diverse groups of people, presenting a significant challenge from a Human-Centric Computer (HCC) perspective. We argue that the semiotic engineering (SemEng) theory, which views human-computer interaction as computer-mediated communication between designers and users at interaction time, may be applied to help creating knowledge in this context. In PaaSE, several designers communicate with PaaSE's users (developers). In this paper, we apply SemEng concepts to analyze different software artifacts present in PaaSE, showing evidence of communication breakdowns between designers and users. Our goal is to provide a better understanding of existing metacommunication processes in such environments, offering specific suggestions to emphasize communication boundaries.**

Keywords*— *semiotic engineering; communicability analysis; cloud computing; platforms as a service (PaaS)

I. Introduction

The constant increasing connectivity and bandwidth availability has enabled the transformation of computing power into a commodity. Companies are gradually adopting the paradigm of cloud computing, aiming at cutting their costs by maintaining data and processing outside their premises. Commonly, cloud computing services fit into three different models that can be viewed as a stack [1]. They are referred to as: Infrastructure as a Service (IaaS), the most basic layer that offers hardware resource abstraction as a service; Platform as a Service (PaaS), which abstracts all resource provisioning, configuration and runtime environment requirements; and at the highest level, Software as a Service (SaaS), where users should deal only with application software, databases and other services, leaving all other aspects to be maintained by the service provider.

The acceptance of cloud computing solutions in the market has grown significantly no matter the level of abstraction of the service model. IaaS solutions, such as the ones from SoftLayer[1] or AWS[2], allow different players, from small business to large enterprises, to shift their operations away from costly hardware maintenance requirements. That is, they allow enterprises to avoid traditional low-level hardware details, so they can focus on their core businesses. However, in these solutions, users still must handle details about networking and provisioning. PaaS solutions, like Amazon Elastic Beanstalk[3], IBM Bluemix[4] and Microsoft Azure[5] aim at hiding these details from their users, creating an abstraction layer that allows them to focus on application development instead of managing their infrastructure. Lastly, SaaS is a model where purchasing and use of software is not related to acquiring licenses, instead, users pay software providers on-demand. SaaS has been incorporated into the strategy of nearly all leading enterprise software companies. Services offered by Salesforce[6] and ServiceNow[7] fall into this category.

In this paper, we focus on PaaS solutions where there is an explicit tradeoff where users (developers) are willing to giving up control of infrastructure specifics in exchange for simplicity and speed. To endorse the positive side of this tradeoff, some PaaS solutions offer users a catalog of services they can choose to integrate with their applications. Commonly, PaaS Environments (PaaSE) must manage information about third-party services (e.g. maintaining documentation with description, features, etc.), references to the service providers, and authentication mechanisms (e.g. API keys, billing, etc.) for contracted services.

The concepts promoted by PaaSE are intrinsically related to the ones from DevOps (Development and Operations). It involves processes and methods for reflection about collaboration and communication between development, IT operations and Quality Assurance staffs. In other words, PaaS promotes a cross-departmental integration, since it depends on heterogeneous expertise to be maintained.

We argue that PaaSE makes an interesting subject for an exploratory study through a communicability perspective. Particularly due to its idiosyncrasies, involving diverse

[1] www.softlayer.com

[2] www.aws.amazon.com
[3] www.aws.amazon.com/elasticbeanstalk
[4] www.ibm.com/bluemix
[5] www.microsoft.com/azure
[6] www.salesforce.com
[7] www.servicenow.com

technical aspects and the collaboration of multiple actors that design and consume different knowledge artifacts. Indeed, some authors [2] state that PaaS is the least mature of the three cloud computing layers usually discussed, but it comes along with a great market potential.

Semiotic Engineering views HCI as a special kind of communication between people, as a computer-mediated communication between designers and users at interaction time [3]. In this sense, the designer of PaaSE has a broad range of responsibilities, since (s)he is consolidating several components, and usually gathering information from other designers of services endpoints or their libraries with APIs. These features must be combined in a "package" to be exposed as one single environment. Among other things, (s)he needs to consistently communicate to his/her users: 1) how this package's features can be organized to build their applications; 2) where and when to "hand over the conversation" to a third-party designer's component or service during users' interaction with the environment.

In this work, we apply concepts from Semiotic Engineering to trace back pitfalls where PaaSE designers typically fail to communicate the user important aspects while building a new application on their platforms. The SigniFYIng Message, a component from the SigniFYI [4] (Signs for Your Interpretation) methodological toolset, was used to frame metacommunication and show breakdowns in the PaaSE's user-designer communication. The Semiotic Engineering view of HCI as communication between designers and users at interaction time provides a distinct perspective for PaaSE evaluation, design and use. We illustrate a fictional PaaSE with interface mockups called "Developer Salad Bar" (DSB), but some of the reported issues were also observed on different PaaSE in the market.

This work is a result of a multidisciplinary group of Semiotic Engineering experts and senior developers. This integration was crucial to enrich analysis and framing the extension of communicability issues in PaaSE. Our main contribution is to provide a better understanding of existing metacommunication processes throughout application development in such environments. In addition, this work provides specific suggestions to emphasize communication boundaries between users (developers) and designers of PaaSE.

This article is organized as follows: the second section presents a background of the problem, with a brief discussion about PaaSE. It also addresses Semiotic Engineering concepts and their framing over PaaS communicability aspects. In the third section, we discuss related works that share interests with our approach. We present a fictional story in the fourth section, showing potential communicability breakdowns between designers and users on PaaSE. In the fifth section, we discuss how the communicability issues presented in our story could be handled with support from Semiotic Engineering tools. In the sixth section, we reflect over communicability aspects on PaaS solutions available in the market, confirming that the theoretical problems addressed in the illustrative scenario are also experienced in practice. Finally, the last section concludes with final remarks and future work.

II. Background

A. PaaS environments

PaaS is the middle tier of the main cloud computing service models, set between IaaS layer (with a lower abstraction level) and SaaS (higher abstraction level). Fig. 1 illustrates the arrangement of these layers with an indication of the main resource they abstract to users, i.e. hardware in the case of IaaS, the platform in PaaSE and applications in SaaS layer.

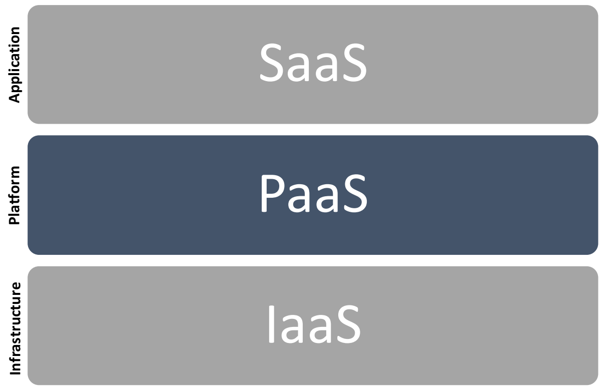


Fig. 1. The three common service models from cloud computing.

NIST [5] provides the following definition for PaaSE, in terms of consumer's capabilities.

"*The capability provided to the consumer* [of PaaSE] *is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.*"

Depending on the desired scenario and restriction policies, there are distinct types of PaaS strategies commonly discussed: public, private, hybrid or community. Public platforms are usually offered to users on the Web, in a pay-as-you-go model. It is also possible to maintain a PaaS in private clusters, restricted to the domain of an organization. Thus, ensuring a finer access control, if security is considered a primary concern. Alternatively, a hybrid model is possible as well. By keeping sensitive data on-premises with restrict access, along with a publicly maintained infrastructure. A fourth strategy is the use of PaaS in a community cloud. That is, the infrastructure is provisioned for a specific community, consumers from one or more organizations. It can be managed by one or more organizations from the community, or by third parties. It can be located on or off premises.

Generally, the PaaS provider is responsible for defining details of how the infrastructure operates, such as its operational system, available programming languages, services, and general management matters. Development tools and other collaborative environments may be provided to enhance users' experience. Fig. 2 summarizes the resources commonly abstracted by PaaSE ranging from hardware (its virtualization and provisioning), along with runtime environments that allow for deployed applications to run.

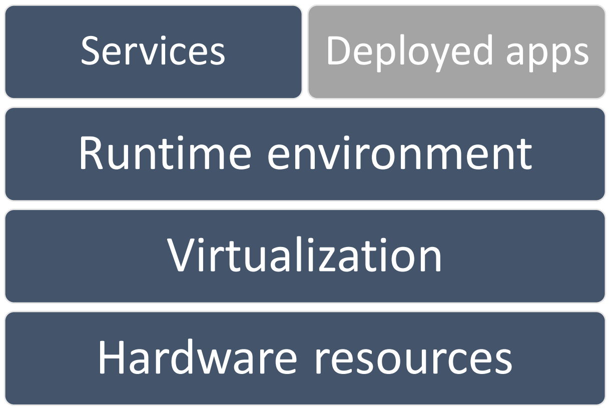


Fig. 2. Resources commonly abstracted by PaaS environments.

The concept of DevOps is an intrinsic ingredient of PaaSE. By promoting the integration of staff from different departments needed for platform management, it is expected to enhance things, e.g., a lower time-to-market, improved product quality, more stable releases, etc. However, this cross-collaboration comes with an associated cost in terms of guaranteeing consistent communication between people. In this sense, an exploratory study with theoretical tools that support epistemological analysis about these communication acts may come in handy.

B. Semiotic Engineering and PaaS Environments

Semiotic Engineering views HCI as a computer-mediated communication act between designers and users at interaction time. The system speaks for its designers in several types of conversations specified at design time. These conversations communicate the designers' understanding of who the users are, what they know the users want or need to do, in which preferred ways, and why [3]. It emphasizes communication and signification processes taking place in interaction, and brings HCI designers onto the stage of human-computer interaction. The system is the designer's proxy during user's interaction (Fig. 3a).

PaaSE have an extra layer of communication in interaction time. PaaSE provide a platform allowing users to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an application [1], [6]. In that environment, there are pieces of software (components, libraries, services, complete applications, etc.) that can be combined to build a new application. The designers of those pieces of software are different people, with different ideas, different goals, different beliefs, and so on. Therefore, while interacting with a PaaSE, the user is communicating with different individuals, depending on which pieces of software (s)he decides to combine (Fig. 3b). The PaaSE's designer is the user's first interlocutor. This designer has the responsibility to intermediate the whole conversation among user and all the other designers of components and services offered by the environment. It is reasonable and expectable that the PaaSE's designer hands over the conversation to other designers throughout the interaction. But this needs to be made clear to PaaSE's users, so they know to whom they are talking with in each step of the interaction.

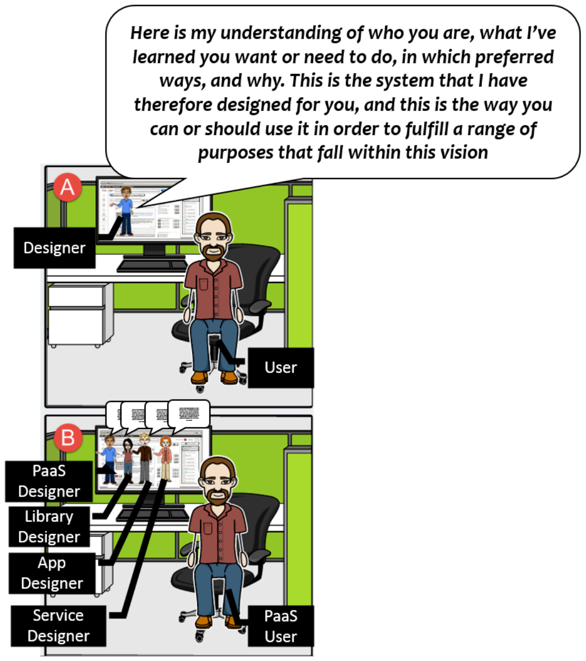

Here is my understanding of who you are, what I've learned you want or need to do, in which preferred ways, and why. This is the system that I have therefore designed for you, and this is the way you can or should use it in order to fulfill a range of purposes that fall within this vision
A
Designer
User
B
PaaS Designer
Library Designer
App Designer
Service Designer
PaaS User

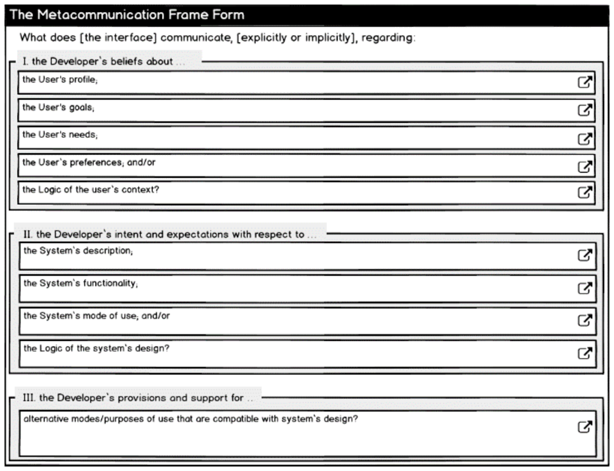
Fig. 3. The system as the designer's proxy (A) and different designers communicating with the user at interaction time (B).

Semiotic Engineering has resources to help addressing this particular situation of PaaSE's designer and multiple designers of combined features (see different designers in Fig. 3b). One interesting resource is the metacommunication template, presented in Fig. 3a dialog balloon. It communicates the designers' understanding of who the users are, what they know the users want or need to do, in which preferred ways, and why. It helps framing and organizing the designers' message, making it easier to identify gaps or confusing portions of that message. SigniFYIng Message is the operational version of the metacommunication template [4]. An evaluator can use the SigniFYIng Message frame, showed in Fig. 4, to define portions of designer-user message:

i. The developer's beliefs about the user's: profile, goals, needs, preferences, and/or the logic of his context;

ii. The developer's intent and expectation with respect to the systems: description, functionality, mode of use, and/or logic of the system's design; and

iii. The developer's provisions and support for: alternative modes/purposes of use that are compatible with system's design.



The Metacommunication Frame Form

What does [the interface] communicate, [explicitly or implicitly], regarding:

I. the Developer's beliefs about ...

- the User's profile;
- the User's goals;
- the User's needs;
- the User's preferences; and/or
- the Logic of the user's context?

II. the Developer's intent and expectations with respect to ...

- the System's description;
- the System's functionality;
- the System's mode of use; and/or
- the Logic of the system's design?

III. the Developer's provisions and support for ..

- alternative modes/purposes of use that are compatible with system's design?

Fig. 4. . Metacommunication Frame Form – SigniFYIng Message frame [4].

The metacommunication template is a known concept from Semiotic Engineering and it has been used as part of the Semiotic Inspection Method [3], [7], [8]. The operational version of it, the SigniFYIng Message, proposes that it can stand on its own as a powerful evaluation resource to identify communicability issues.

We applied the SigniFYIng Message to frame a portion of the PaaSE designer's message and to expose this environment's particularities, with multiple designers communicating with one user through "mediation" of the PaaSE's designer.

III. Related Work

There is few previous research related to human-centered aspects of PaaSE. The work presented in [9] discusses about customer satisfaction measures by considering a user feedback mechanism based on Interview and Questionnaire method, which is the closest it gets to an HCI result. It also evaluates performance by taking CPU utilization, memory usage and disk seeking rate as essential parameters. Some previous research was made regarding cloud interaction in general [10] or about cloud systems evaluation all together, considering issues that are transparent to PaaSE's users like infrastructure, for example [11], [12].

To the best of our knowledge, there is no proper HCI evaluation of PaaSE that:

- Provides to the users the capability to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider; and
- Provides resources so the user does not need to manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment

The first point is an important one in our perspective, when all the interaction between user and PaaSE's interface happens: when the user wants to build a new application, and uses the PaaSE to achieve that goal. PaaSE's designer needs to define the boundaries between the message (s)he is responsible for and the message from other designers (library APIs, RESTful service endpoints, apps, etc.).

Collaborative systems research is a field where there are multiple people communicating to achieve a goal. However, for collaborative systems, there are multiple users, not multiple designers, as our PaaSE scenario. Semiotic Engineering methods were previously used to evaluate collaborative systems and some results might be interesting for our investigation. The need of discourse and coordination awareness for collaborative systems, considering multiple people communicating [13], could be applied to our scenario. However, as aforementioned, PaaSE have one user and multiple designers. We are particularly interested in the discussion of following two concepts.

Discourse awareness: PaaSE's designer needs to be aware of the other designers' (library APIs, service endpoints, and other designers related to application DevOps) discourses and make users aware of each discourse. Users must always be able to identify to which designer (s)he is communicating while interacting with the PaaSE.

Coordination awareness: PaaSE's designer needs to coordinate with other designers to define the boundaries of information each one is going to communicate to users and how. PaaSE should not be responsible for communicating all information about library APIs, service endpoints, etc., but there is a border that needs to be defined to avoid PaaSE's users to experience a communicability breakdown in interaction time, which includes design, development and deployment stages.

IV. Scenario

The following fictional scenario, based on true facts that happened with experienced developers, illustrates how tricky the communicability between PaaS designers and users (application developers) of these systems can be.

John is a senior developer working as a manager at a software company with an established portfolio on building both web and stand-alone applications. One day, a client orders the development of a new e-commerce website that should be implemented in a scalable way. That is, it should initially meet the demand of hundreds of costumers, but potentially serving up to a "couple of thousands" of consumers simultaneously. In addition, the client wants the system to support different data analysis, such as consumer profiling and sales statistics.

Although John did not have experience with cloud development before, he promptly realizes that this new project could benefit from the much-advertised "elastic" hosting scheme these systems offer. He also thinks this is an opportunity to bring his company's portfolio to another level. John knows there are different cloud-based platforms available, which are capable of rapidly allocating computational resources depending on the actual access demand and workload. Furthermore, some of these platforms provide off-the-shelf services with analytic algorithms that support big data exploration. He decides that it would be a clever idea to prototype a sample application in one of the PaaS solutions available in the market. Through this prototype, it would be easier to talk about implementation possibilities and the client's requirements.

Along with his team, John initiates the development of the prototype using tools provided by a well-known PaaS corporation, named Developer Salad Bar (DSB). They start by defining what services the prototype needs and initially identify three building blocks it should have: an HTTP server technology to host the e-commerce website, a database service to store information about customers and products, and a consumer behavior analysis service to support reasoning about statistics and business trends. Luckily, all these services are offered by the DSB platform. Furthermore, it provides ready-for-use boilerplate code in a variety of programming languages, to help users overcoming the initial learning curve to deal with the platform specifics.

After logging in the DSB's website, John's team were presented with an administration interface showing a list of several programming languages and execution environment options that they could choose for their projects, as well as a service collection they could use. Fig. 5 illustrates the interface for project configuration offered to John's team while using the DSB platform.

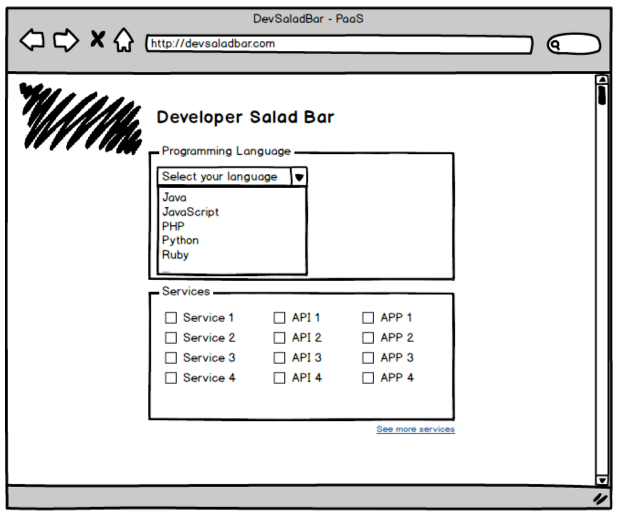


Fig. 5. . DSB's administration interface.

DSB offers a broad range of services, including some based on machine learning techniques, including computer vision and data analytics algorithms. For each service available, there is an associated documentation describing its features in detail. John goes on to read the documentation provided by DSB and, serendipitously, a specific feature in a data analytics service calls his attention: a trend analysis component to support exploring consumers' personality and habits. By tracking consumers' interaction events and input data, the service can estimate whether they are prone to acquire and experiment new products, or they have a more conservative profile, preferring to keep their habits and acquire the products they customarily get. Fig. 6 depicts the webpage with the service's primary features.

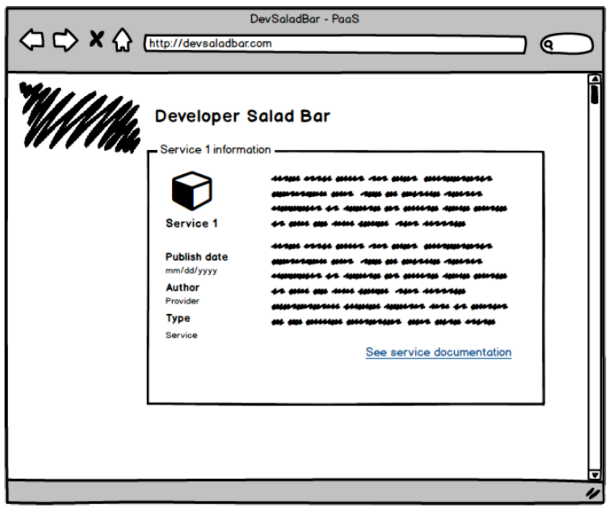


Fig. 6. . Service description page in DSB.

John also learns from the DSB's documentation that the platform automatically configures the execution environment based on user's choice of programming language and services. That is, after the application code is deployed, it performs a transparent provisioning of resources and fetching procedures to handle all software dependencies. These software dependencies include the associated libraries that provide access to selected services, in the programming language defined by the user. In other words, the platform takes care of the whole configuration process and application execution, in a hassle-free approach.

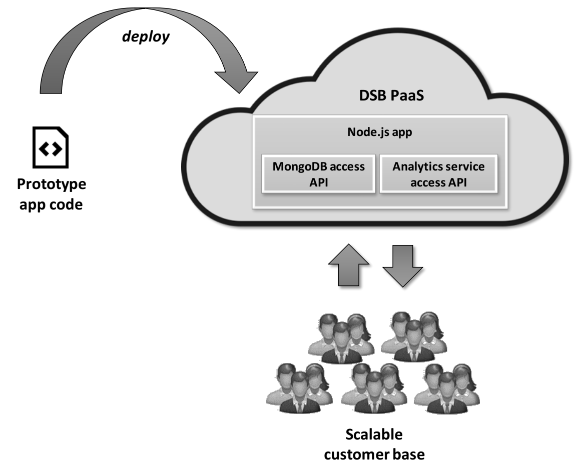


Fig. 7. Sketch with prototype overview created by John and his team.

After a comprehensive meeting discussing technology choices, John and his team define they will stick with software tools they had previous experience. They opt to work with JavaScript language on the server side, specifically with

Node.js runtime environment for the HTTP server. For the database technology, the team chooses MongoDB, a NoSQL database. They outline an overview of the prototype's architecture (Fig. 7), and get satisfied with the overall simplicity of the project as a result of the various features offered by the DSB platform.

Designers in the team start to create the website visual identity, while database experts discuss how they will model the database considering the client's business requirements. After some work effort, the team manages to establish a functional prototype with basic HTTP server and database components. Therefore, they are ready to begin experiment with some of the features provided by the platform.

John suggests the team should try out the consumer analytics features he saw in DSB's documentation. But to everyone's surprise, after all the preparation on the prototype's database and interface to receive and present results from the trend analysis component, the specific calls to the features that John saw in DSB's documentation are not present in the provided Node.js library, which abstracts the access to the analytics RESTful service endpoints.

John refers again to the documentation and indeed finds the remote calls to the features he wants. But he realizes something that had been overlooked. The link to the documentation actually leads to an external service website. After clicking that link (displayed in the DSB's service page in Fig. 6), the user is taken to the service's developer specific documentation, as can be seen in the browser's address bar in Fig. 8.

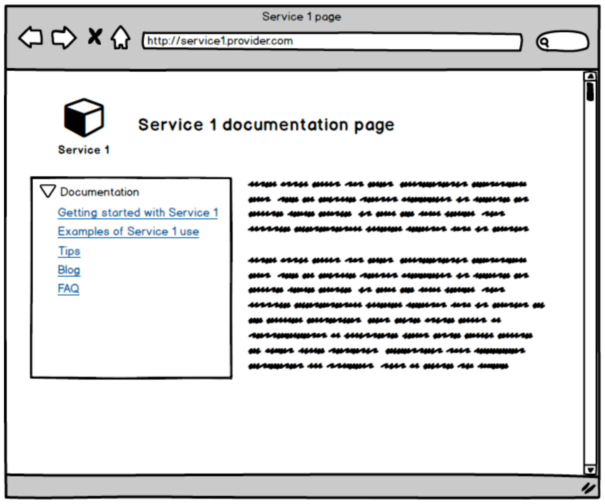

Service 1 page
http://service1.provider.com
Service 1
Service 1 documentation page
Documentation
Getting started with Service 1
Examples of Service 1 use
Tips
Blog
FAQ

Fig. 8. Service-specific external documentation.

Unsure what to do, John contacts the DSB's support team and describes the situation. They explain that the platform offers service endpoints (in a RESTful scheme), which can be abstracted by third-party libraries. In this case, there must be some inconsistency between the features offered by the service and the access to endpoints available from the service library. Moreover, John is told that their execution and request processing take place outside of DSB's premises. That is, remote calls from the final users' front-end to these endpoints are forwarded to exogenous systems that handle the requests accordingly.

John and his team review their draft architecture considering the information received from the platform support team. Fig. 9 shows the platform details they were not aware during the initial prototype development.

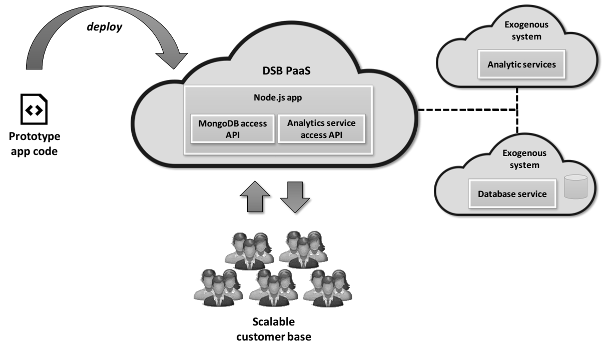


Fig. 9. Prototype overview considering details received from DSB's support team.

There is a clear indirection regarding remote requests coming from users on the Web to the platform services. Requests or calls are received through libraries in the server module and dispatched to the respective service implementations, which may reside outside of the platform premises. These exogenous systems are responsible for processing requests and returning results of the performed computing to the HTTP server in PaaS, which in turn may return them to clients who originated the request.

After investing significant person-hours in the prototype, John and his team are facing a tricky situation and need to make a decision: They can either give up using their favored programming language (JavaScript) and search for another one which has a service library consistent with the latest features offered by the service. Or, they can invest more effort in the current setup, trying to work around the problem by extending the library offered by the platform, or even rework it completely from scratch. Either way, John and his team will need to invest more time that they assumed they would have, since the PaaS approach should have facilitated their development, but ended up in this turmoil. If they had all the information beforehand, they might have taken another way to prototype the solution.

V. Communicability Analysis

We performed a communicability analysis and discussion with the participation of experts in Semiotic Engineering and HCI, and experienced developers. First, a Semiotic Engineering and HCI expert filled the SigniFYIng Message frame (Fig. 4), considering the DSB's interface that the user selects the programming language and the services that (s)he wants to combine in his/her application (Fig. 5):

i. **The developer's beliefs about the users**:

You are a developer who needs to combine languages with services, APIs and APPs to build applications...

ii. **The developer's intent and expectation with respect to the system**:

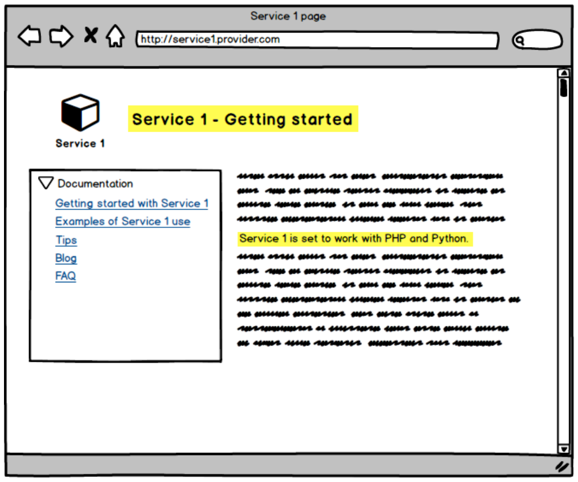
Therefore, here you can choose ANY LANGUAGE and combine it with ANY SERVICES, APIs or APPs that you wish…

iii. **The developer's provisions and support for**:

…to build your new application for which I will automatically configure the execution environment based on your choices of programming language and services.

In Fig. 6, the service's documentation does not contradict the metacommunication message, so the user thinks (s)he is ok to go with his/her choices of programming language and service to build his/her new application.

Once the development advances, John and his team realize that they cannot easily use the service they needed (consumer behavior analysis service) with the language they wanted (JavaScript). John goes back into the service documentation page (Fig. 6) and notices a link to the service's provider documentation (Fig. 8), outside DSB's domains. At this point, John gave up and went to look for helpdesk assistance, but the service's provider documentation page would surprise him as well by contradicting the DSB's interface, once it informs the user that the service only works with PHP and Python, not JavaScript (Fig. 10).


Service 1 page

http://service1.provider.com

Service 1

Service 1 - Getting started

Documentation

Getting started with Service 1

Examples of Service 1 use

Tips

Blog

FAQ

Service 1 is set to work with PHP and Python.

Fig. 10. Service 1 - Getting started page.

Considering the portion of interaction that ended in Fig. 10, we identified a contradiction between the message sent by the DSB's designer, presented in the first SigniFYIng Message frame and the message sent by the service provider's designer in his "Getting started" page:

i. **The developer's beliefs about the users**:

You are a developer who needs to combine languages with services, APIs and APPs to build applications…

ii. **The developer's intent and expectation with respect to the system**:

However, I ONLY PROVIDE DIRECT COMBINATION WITH PHP AND PYTHON…

iii. **The developer's provisions and support for**:

Therefore, to build your new application with my service you need to know PHP or PYTHON, otherwise you are on your own.

The absence of that crucial information in the DSB's interface (programming languages that a service is ready to be combined) misled John on to think that there was no restriction regarding programming languages for the service he wanted to use in his application. Therefore, he and his team went over with their plans to build the application with the service they needed. When they found out that they could not use the service with the language they were using to program the new application, some rework would be necessary no matter the decision on that point, and they would spend more time and money to adjust to the new scenario. The facilities that the PaaSE offered at first could be evaluated if they knew about the programming language limitation of that service.

Regarding the expected calls not provided in the Node.js library, there is an inconsistency of information between the DSB's documentation and the provided Node.js library. John went to the documentation and indeed found the calls to the features he wanted. However, the reference to an external site (the service's provider documentation site) made him doubt the PaaSE designer's message learned in the documentation (Fig. 6). Therefore, he looked for the information outside the interaction space, calling the DSB's support team.

i. **The developer's beliefs about the users**:

You are a developer who needs calls to a specific feature of a service.

ii. **The developer's intent and expectation with respect to the system**:

Here are the remote calls to the features you want …

iii. **The developer's provisions and support for**:

…so, you can integrate with your new application.

However, when John went over the Node.js library that abstracts the service endpoints, he did not find the call that he was expecting to find. In this library, we frame the message from its designer to users (developers). In that message, the user did not find the remote call to the features he was expecting. The DSB's support team explained to John that the platform offers service endpoints (in a RESTful scheme), which can be abstracted by third-party libraries. However, he did not anticipate having to deal with this kind of things; he expected that DSB would do that for him.

The SigniFYIng Message frames filled above are messages from different designers: PaaSE's designer and service's designer. The user gets lost on that communication. The PaaSE's designer do not inform the user his/her "boundaries of communication" during interaction. The user is sent to another website (the service's provider documentation website) without further notice, which in this case caused communication breakdowns in the interaction for PaaSE's users (Fig. 11).

It is important to stress that, considering calls to service's endpoints, there is an inconsistency problem, which also leads to communication breakdowns. PaaSE's designer says one thing, but the service's designer says another thing and misleads users. In the scenario where John contacts the DSB's support team, there are even more evidences that PaaSE's behavior might be too "transparent" to users. Maybe users need to choose how "in the dark" they want to be about their application implementation details.

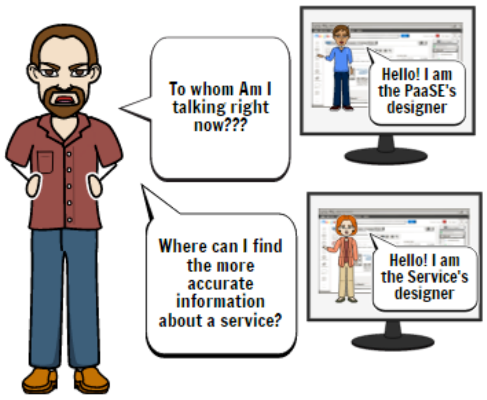

To whom Am I talking right now???
Hello! I am the PaaSE's designer
Where can I find the more accurate information about a service?
Hello! I am the Service's designer

Fig. 11. Not so happy PaaSE user.

The PaaSE show some information about the service (Fig. 6), but the programming languages and the libraries that abstract the endpoints of services are not part of that set of information. This communicability analysis indicated how crucial this information is and needs to be part of the communication between PaaSE's designer and his/her users (developers) about services in advance.

In our communicability analysis, we did not perform a complete evaluation of DSB, so there is no distinction among types of signs or complete filled metacommunication templates or comparisons, as expected in a detailed semiotic inspection [3], [7]. We applied SigniFYIng Message to frame the metacommunication message of different designers of a PaaS environment to show how it can be unclear to its users.

Our analysis shows evidences that PaaSE's designer needs to be aware of the other designers' discourses (present in library APIs, service endpoints, and other designers related to application DevOps) and make the user aware of each discourse. The user must always know to which designer (s)he is communicating with, while interacting with PaaSE (Discourse awareness). In addition, PaaSE's designer needs to coordinate with other designers to define the boundaries of information each one is going to communicate to users and how this communication would happen. PaaSE's designer should not be responsible for communicating all information about libraries, services, etc., but there is a border that needs to be defined to avoid users to experience communicability breakdowns (Coordination awareness).

VI. PaaSE Solutions in Practice

There are many PaaS solutions available today, each one applying different strategies for communicating their intended use and development abstractions, i.e. execution platforms, available services and libraries. In this sense, we discuss and reflect about general communicability aspects of three representative platforms that are currently widely used: Amazon Elastic Beanstalk, Microsoft Azure, and IBM Bluemix. The inspection described in this section is by no means exhaustive, but rather, it illustrates some difficulties and communicative problems that developers commonly face when using PaaSE and its many features, even with all the facilities these solutions provide. The three inspected platforms offer a substantial number (see Table 1, at the end of this section) of features. To demonstrate the developers' experience of selecting services and runtime environments for their applications, we conducted a brief inspection to explore how these PaaSE classify and present their features to users. Table 1 summarizes the categories and total number of services and runtime environments available in each one. The runtime environments supported by the three platforms are mostly similar, with some exceptions. All of them support software development in a common subset of languages such as Python, Java, PHP, JavaScript (through Node.js), and Ruby.

Currently, Amazon Elastic Beanstalk offers a total of 105 services distributed in 19 categories. The platform documentation[8] states that applications can integrate any of the offered AWS services, which are not managed in the user's environment provided by Elastic Beanstalk. The architecture overview suggests that applications and services typically work across multiple "availability zones". It does not specify whether there is any service hosted on third-party infrastructure. Fig. 12 shows the dashboard listing the diverse services available for developers using the platform. Screenshots of the analyzed platforms in this section were anonymized to avoid publication issues with registered trademarks and logos.

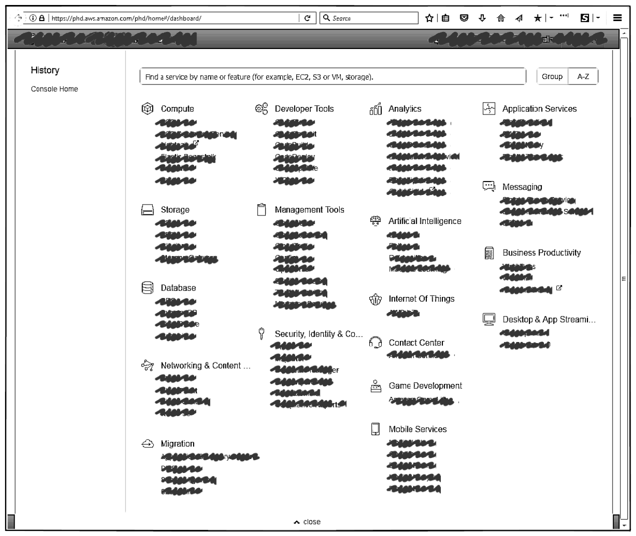
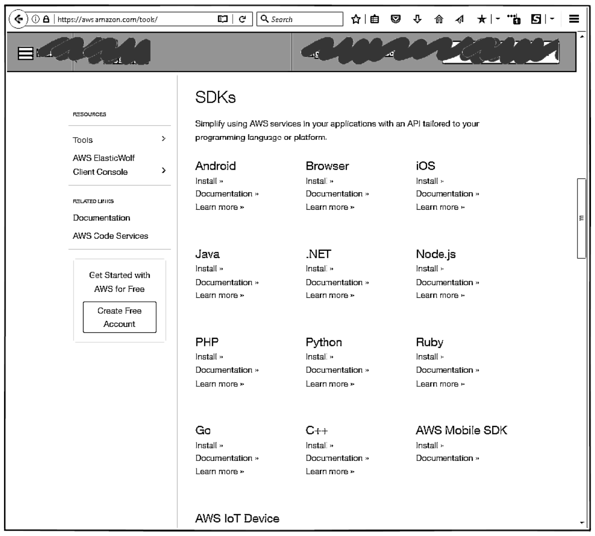


Fig. 12. Amazon Elastic Beanstalk dashboard with a listing of available services.

[8] http://docs.aws.amazon.com/elasticbeanstalk/



https://aws.amazon.com/tools/

[redacted] [redacted]

RESOURCES

Tools >

AWS ElasticWolf Client Console >

RELATED LINKS

Documentation

AWS Code Services

Get Started with AWS for Free

Create Free Account

SDKs

Simplify using AWS services in your applications with an API tailored to your programming language or platform.

Android	Browser	iOS
Install »	Install »	Install »
Documentation »	Documentation »	Documentation »
Learn more »	Learn more »	Learn more »

Java	.NET	Node.js
Install »	Install »	Install »
Documentation »	Documentation »	Documentation »
Learn more »	Learn more »	Learn more »

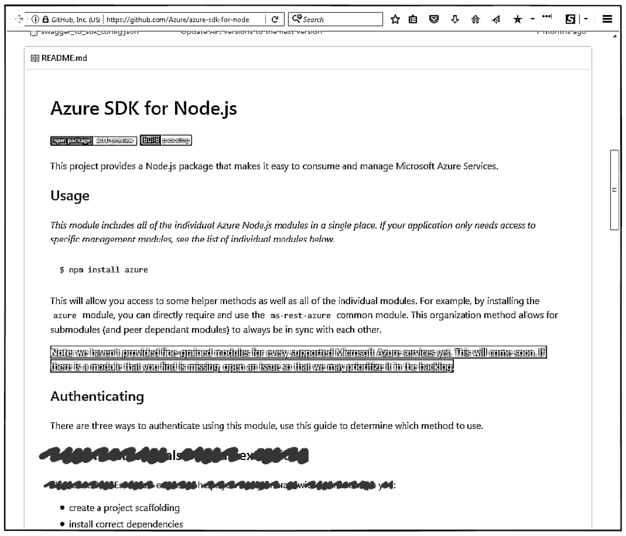
PHP	Python	Ruby
Install »	Install »	Install »
Documentation »	Documentation »	Documentation »
Learn more »	Learn more »	Learn more »

Go	C++	AWS Mobile SDK
Install »	Install »	Install »
Documentation »	Documentation »	Documentation »
Learn more »	Learn more »	

AWS IoT Device

Fig. 13. Provided SDKs to use AWS services in all supported runtime environments.

Considering software development artifacts, PaaSE usually provide APIs in the form of libraries or software development kits (SDKs) to facilitate access to their services. Fig. 13 shows the options offered by Amazon Elastic Beanstalk for its supported runtimes. For each of the available programming languages and platforms there are links to release packages, documentation with API references and GitHub repositories with source code maintained by open-source communities.



GitHub, Inc. (US) https://github.com/Azure/azure-sdk-for-node

README.md

Azure SDK for Node.js

npm package build passing

This project provides a Node.js package that makes it easy to consume and manage Microsoft Azure Services.

Usage

This module includes all of the individual Azure Node.js modules in a single place. If your application only needs access to specific management modules, see the list of individual modules below.

```
$ npm install azure
```

This will allow you access to some helper methods as well as all of the individual modules. For example, by installing the `azure` module, you can directly require and use the `ms-rest-azure` common module. This organization method allows for submodules (and peer dependant modules) to always be in sync with each other.

Note: we haven't provided fine-grained modules for every supported Microsoft Azure services yet. This will come soon. If there is a module that you find is missing, open an issue so that we may prioritize it in the backlog.

Authenticating

There are three ways to authenticate using this module, use this guide to determine which method to use.

[redacted]

[redacted]:

- create a project scaffolding
- install correct dependencies

Fig. 14. List of supported AWS services (and specific API version) of the provided Node.js SDK.

Each of the provided SDKs has specificities on how to communicate the supported services and their API versions to developers. The supported services may vary as well. For instance, the Node.js SDK shows a list[9] of exactly 100 supported services (specified in a SERVICES.md file in its current master repository, see Fig. 14), while the Java SDK sums up 88 services (as per its release notes[10] in version 1.11.82).

Microsoft Azure offers a total of 101 services across 11 product categories (cf. Table 1). These services are developed by Microsoft itself. Unlike Elastic Beanstalk, the Microsoft solution offers numerous third-party services. The developers and maintainers of offered services are clearly listed in a dashboard area called Marketplace (see Fig. 15).

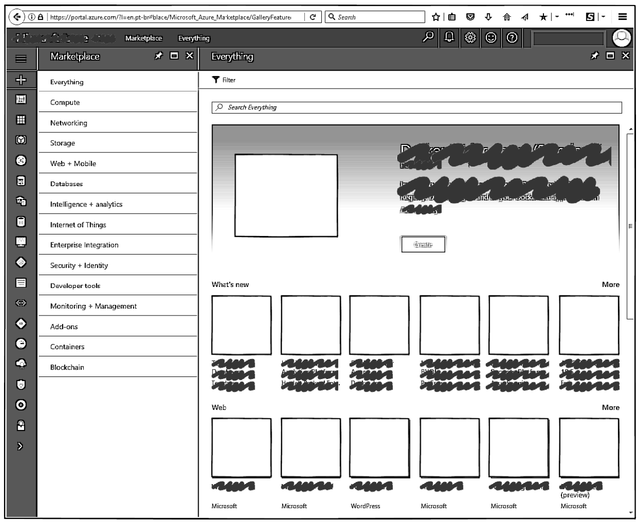


Fig. 15. Microsoft Azure dashboard with a listing of available services and other products.

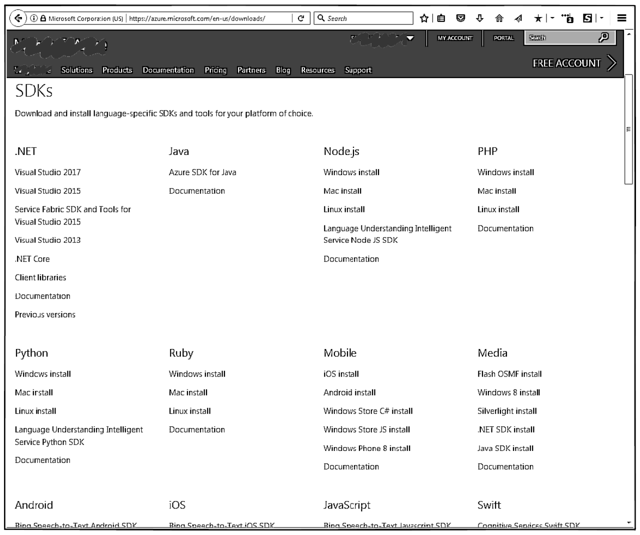
Microsoft's approach is to make developers' experience of composing services and using platform features into something analogous to choosing and buying software products from an app store. This facilitates third-party developers (or publishers in Azure terms) making their services promptly available, increasing the product offer on the platform. Which on the one hand is good, given that the solution today lists approximately 600 products available[11]. On the other hand, makes attaining a communicative cohesion among the various designers and consumers of artifacts of the platform even more complex.

Fig. 16 shows the SDK listing for accessing Microsoft's services, which are made available through repositories in GitHub under open-source license. Each community manages the service SDK lifecycle in its own way. Taking the Node.js SDK repository for reference again, its README.md file (which serves as the GitHub repository webpage) states the following (see highlighted text in Fig. 17): "Note: we have not provided fine-grained modules for every supported Microsoft Azure services yet. This will come soon. If there is a module that you find is missing, open an issue so that we can prioritize it in the backlog". Certainly, the current implementation covers most of the platform services. Nevertheless, this could be a potential source of a communication mismatch, since this information is not explicitly available on the PaaSE's page.

[9] https://github.com/aws/aws-sdk-js/blob/master/SERVICES.md

[10] https://aws.amazon.com/releasenotes/Java/6001466639362819

[11] https://azure.microsoft.com/en-us/services/


Microsoft Corporation (US) | https://azure.microsoft.com/en-us/downloads/

[redacted] [redacted] MY ACCOUNT PORTAL Search

[redacted] Solutions Products Documentation Pricing Partners Blog Resources Support FREE ACCOUNT

SDKs

Download and install language-specific SDKs and tools for your platform of choice.

.NET
Visual Studio 2017
Visual Studio 2015
Service Fabric SDK and Tools for Visual Studio 2015
Visual Studio 2013
.NET Core
Client libraries
Documentation
Previous versions

Java
Azure SDK for Java
Documentation

Node.js
Windows install
Mac install
Linux install
Language Understanding Intelligent Service Node JS SDK
Documentation

PHP
Windows install
Mac install
Linux install
Documentation

Python
Windows install
Mac install
Linux install
Language Understanding Intelligent Service Python SDK
Documentation

Ruby
Windows install
Mac install
Linux install
Documentation

Mobile
iOS install
Android install
Windows Store C# install
Windows Store JS install
Windows Phone 8 install
Documentation

Media
Flash OSMF install
Windows 8 install
Silverlight install
.NET SDK install
Java SDK install
Documentation

Android
Bing Speech-to-Text Android SDK

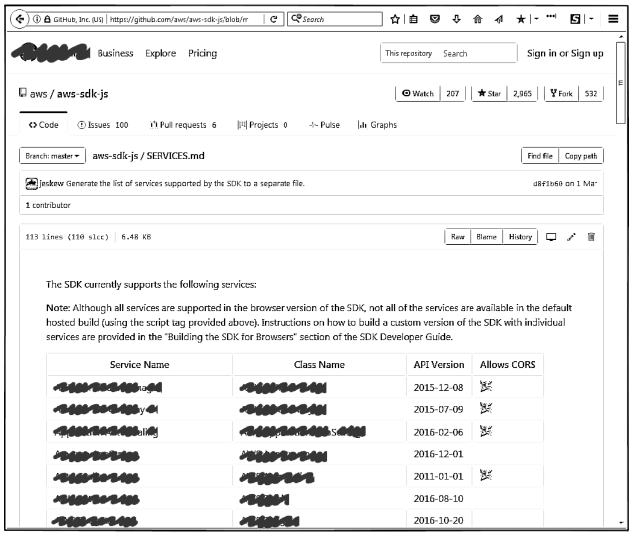
iOS
Bing Speech-to-Text iOS SDK

JavaScript
Bing Speech-to-Text Javascript SDK

Swift
Cognitive Services Swift SDK

Fig. 16. Provided SDKs to use Microsoft Azure services in supported runtime environments..

IBM Bluemix currently has 119 services in 11 distinct categories, of which 44 are developed and maintained by third parties. Fig. 18 shows part of its catalog listing, where tags are used to highlight whether artifacts such as services, boilerplates for apps and runtime environments are maintained by third-party enterprises and open-source communities.


GitHub, Inc. (US) | https://github.com/aws/aws-sdk-js/blob/m

[redacted] Business Explore Pricing This repository Search Sign in or Sign up

aws / aws-sdk-js Watch 207 Star 2,965 Fork 532

Code Issues 100 Pull requests 6 Projects 0 Pulse Graphs

Branch: master aws-sdk-js / SERVICES.md Find file Copy path

jeskew Generate the list of services supported by the SDK to a separate file. d8f1b60 on 1 Mar

1 contributor

113 lines (110 sloc) | 6.48 KB Raw Blame History

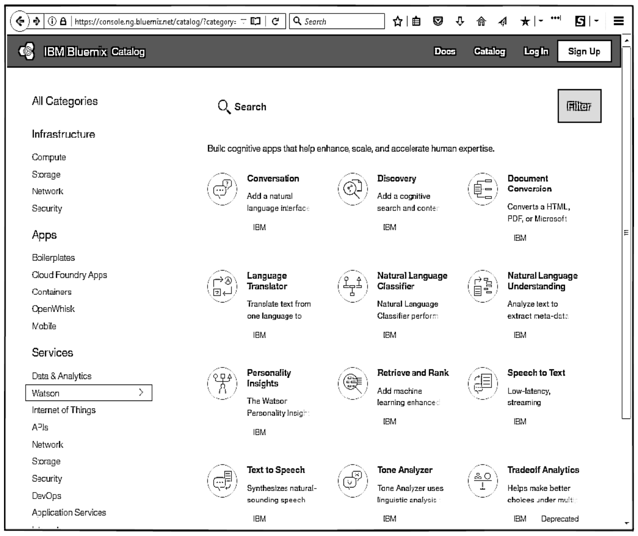
The SDK currently supports the following services:

Note: Although all services are supported in the browser version of the SDK, not all of the services are available in the default hosted build (using the script tag provided above). Instructions on how to build a custom version of the SDK with individual services are provided in the "Building the SDK for Browsers" section of the SDK Developer Guide.

Service Name	Class Name	API Version	Allows CORS
[redacted]	[redacted]	2015-12-08	✓
[redacted]	[redacted]	2015-07-09	✓
[redacted]	[redacted]	2016-02-06	✓
[redacted]	[redacted]	2016-12-01	
[redacted]	[redacted]	2011-01-01	✓
[redacted]	[redacted]	2016-08-10	
[redacted]	[redacted]	2016-10-20	

Fig. 17. Source-code repository for Node.js SDK with a note stating that some services may be missing.

Differently from the other two discussed solutions, the Bluemix platform does not aim at providing a single SDK that incorporates facilities for accessing all services. Instead, there is a different approach depending on the specific feature. For instance, the platform offers different SDKs to for using Internet of Things and Watson services. In another specific case, for accessing the Cloudant NoSQL storage service, IBM provides a specific library rather than an SDK with support for multiple storage services.


https://console.ng.bluemix.net/catalog/?category=

IBM Bluemix Catalog Docs Catalog Log In Sign Up

All Categories
Infrastructure
Compute
Storage
Network
Security
Apps
Boilerplates
Cloud Foundry Apps
Containers
OpenWhisk
Mobile
Services
Data & Analytics
Watson
Internet of Things
APIs
Network
Storage
Security
DevOps
Application Services

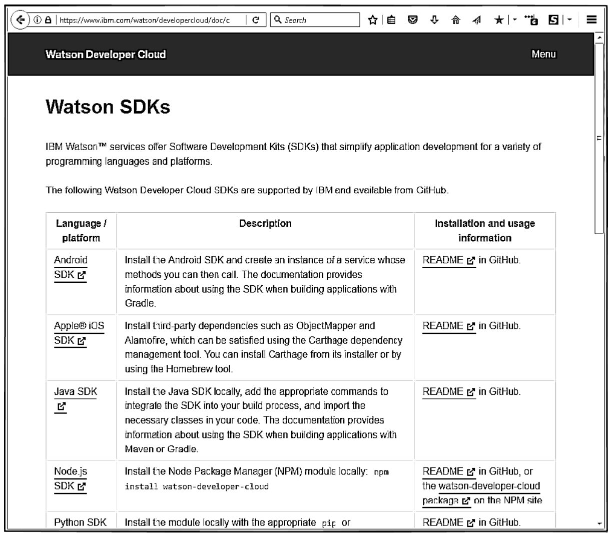
Search Filter

Build cognitive apps that help enhance, scale, and accelerate human expertise.

Conversation — Add a natural language interfac — IBM
Discovery — Add a cognitive search and conte — IBM
Document Conversion — Converts a HTML, PDF, or Microsoft — IBM
Language Translator — Translate text from one language to — IBM
Natural Language Classifier — Natural Language Classifier perform — IBM
Natural Language Understanding — Analyze text to extract meta-data — IBM
Personality Insights — The Watson Personality Insigh — IBM
Retrieve and Rank — Add machine learning enhance — IBM
Speech to Text — Low-latency, streaming — IBM
Text to Speech — Synthesizes natural-sounding speech — IBM
Tone Analyzer — Tone Analyzer uses linguistic analysis — IBM
Tradeoff Analytics — Helps make better choices under mult — IBM Deprecated

Fig. 18. IBM Bluemix services listing, with tags identifying third party, community and IBM maintained components.

Fig. 19 shows the listing of Watson SDKs for the different supported runtime environments, specifically for handling calls to Watson services. In this specific listing, there is the following observation: "Some services are not available with all SDKs, and some services support additional SDKs. See the documentation for the service you want to work with for more information". This again points to a potential breakdown in communication that developers (users) may come across.


https://www.ibm.com/watson/developercloud/doc/c

Watson Developer Cloud Menu

Watson SDKs

IBM Watson™ services offer Software Development Kits (SDKs) that simplify application development for a variety of programming languages and platforms.

The following Watson Developer Cloud SDKs are supported by IBM and available from GitHub.

Language / platform	Description	Installation and usage information
Android SDK	Install the Android SDK and create an instance of a service whose methods you can then call. The documentation provides information about using the SDK when building applications with Gradle.	README in GitHub.
Apple® iOS SDK	Install third-party dependencies such as ObjectMapper and Alamofire, which can be satisfied using the Carthage dependency management tool. You can install Carthage from its installer or by using the Homebrew tool.	README in GitHub.
Java SDK	Install the Java SDK locally, add the appropriate commands to integrate the SDK into your build process, and import the necessary classes in your code. The documentation provides information about using the SDK when building applications with Maven or Gradle.	README in GitHub.
Node.js SDK	Install the Node Package Manager (NPM) module locally: `npm install watson-developer-cloud`	README in GitHub, or the watson-developer-cloud package on the NPM site
Python SDK	Install the module locally with the appropriate `pip` or	README in GitHub.

Fig. 19. IBM Bluemix provides separated SDKs for groups of offered services, such as for Watson APIs.

In general terms, all three solutions consistently organize their offered features, grouping them by similarity and use scenarios. They also provide reasonably adequate documentation. However, the resulting combination from choosing among runtime environments, programming languages, libraries, and desired services is not always smooth and guaranteed as developers may think.

TABLE I. OVERVIEW OF THE INSPECTED PAAS SOLUTIONS

PaaS environment	Category	Services	Runtime environments
Amazon Elastic Beanstalk	Compute	10	Java, PHP, .NET, Node.js, Python, C++, Go, Objective-C, and Ruby
	Storage	8	
	Database	7	
	Migration	6	
	Networking & Content Delivery	5	
	Developer Tools	7	
	Management Tools	12	
	Artificial Intelligence	4	
	Analytics	9	
	Security, Identity & Compliance	11	
	Mobile Services	6	
	Application Services	4	
	Messaging	4	
	Business Productivity	3	
	Desktop & App Streaming	2	
	Software	1	
	Internet of Things	3	
	Contact Center	1	
	Game Development	2	
Microsoft Azure	Compute	10	.NET, Java, Python, PHP, Ruby, Node.js
	Networking	9	
	Storage	9	
	Web + Mobile	13	
	Databases	6	
	Intelligence + Analytics	26	
	Internet of Things	4	
	Enterprise Integration	2	
	Security + Identity	6	
	Developers Tools	4	
	Monitoring + Management	12	
IBM Bluemix	Data & Analytics	35	Liberty for Java, SDK for Node.js, ASP.NET Core, Runtime for Swift, Xpages, Go, PHP, Python, Ruby, Java Tomcat
	Watson	13	
	Internet of Things	11	
	API Management	1	
	Network	1	
	Storage	1	
	Security	7	
	DevOps	15	
	Application Services	30	
	Integrate	5	

Finally, the observed scenario of ever increasing number of available features, services, and tools, envisions a challenging setting in terms of effective communication between the various PaaSE interlocutors. Finding certain information in the existing data deluge can be a nontrivial task.

VII. FINAL REMARKS

In this paper, we discuss how existing metacommunication processes in PaaS application development could be better understood and improved, through the communicability perspective provided by the Semiotic Engineering's theory. The PaaS abstraction model naturally entails the involvement of different designers and users through different software and knowledge artifacts, which makes it an interesting object of study.

A first contribution of this paper is to identify opportunities for improving communicability on PaaSE (more highlighted information about the relation of programming languages/libraries), which could have prevented a lot of unnecessary work from John and his team. This is an example of how the SigniFYIng Message tool could be applied to pinpoint communicability breakdowns. Semiotic Engineering has more resources to help on that matter, including other tools from the SigniFYI (Signs for Your Interpretation) suite [4].

A second contribution that is worth mentioning is bringing collaborative aspects of communication to the PaaS environment (e.g. "user ←→ platform designer", "user ←→ service designer", "platform designer ←→ service designer", and "user ←→ library designer" relations). This inspired us to apply concepts from Collaborative Systems [13], such as discourse awareness and coordination awareness. We believe that these concepts can guide further investigations in PaaSE communicability analyses.

A third contribution of this work is to bring a multidisciplinary perspective to the semiotic inspection context. Indeed, our findings are a result of discussions and reflections by a group of Semiotic Engineering experts and experienced developers. With a multidisciplinary team, we were able to discuss communicability issues in a deeper level when compared to traditional HCI approaches. The discussed scenario was based on true facts experienced by developers with theoretical and practice expertise. They found a workaround solution for the PaaSE communicability issue. But how about other developers? Would they be able to find their way around it? We believe PaaSE designers should help them to fulfill their goals of developing an application in a facilitated way by consistently communicating their intents over platform features and how to use them.

A final contribution of this work is the reflection over communicability issues on three representative solutions available in the market. Through a succinct analysis it was possible to identify breakdowns in the communicability of such platforms, confirming that the analysis performed on the fictional story is observed in practice as well. A survey on the services offered by these solutions illustrates the substantial number of features, and hence the complexity involved in the process of modeling and defining system requirements developed on top of the three platforms.

PaaSE's main goal is to offer abstractions for application programming in the cloud. However, the greater the volume of offered resources, the greater is the number of actors participating in the design and use of such resources, making communication among them more complex. As seen in the brief overview of widely used solutions, the combination of offered features may not be as seamless as the PaaSE designers' message may appear to users. From the designers' point of view, one simplistic way to mitigate communicability issues would be not to offer abstractions for using platform services in the supported runtime environments, making catalogs of services and libraries external to PaaSE. Obviously, this clearly contrasts with PaaSE's inherent purposes. Another way of handling such issues would be for PaaSEs not to present any catalog of services before users select their desired runtime environment. In this case, only the services actually supported in that specific programming language and runtime would be presented. Note, however, it would be necessary to specify mechanisms for matching services offered by the PaaSEs and their support by the existing libraries. Not to mention the marketing issues that this alternative would entail, since the offered services would not be presented beforehand. Finally, a further alternative to address communicability problems when integrating new services to a specific PaaSE is to require a clear message from their designers about which service abstractions are offered for each runtime environment. Thus, avoiding communicability breakdowns, and most important, frustration from PaaSEs users.

We believe that by shedding light on some of the problems discussed in this work, we contribute to a more comprehensive assessment of communicability issues in the context of software development processes in PaaSE. In this sense, we hope to incrementally contribute to a better interpretation of human-centered aspects that are closely related to Software Engineering in these environments.

We envision a complete communicability evaluation on an actual application developed on top of a PaaS environment as a future work. In addition, addressing the identification and inspection of designer-user metacommunication throughout the processes of design, development and application deployment in PaaSE. We also intend to further explore the SigniFYI methodological toolset [4], to assist us uncovering meanings inscribed in software artifacts in such environments. With this suite, we could investigate metacommunication and their perceived effects in a holistic approach, observing its propagation during the software development stages in PaaSE. One of the components in the suite is particularly interesting to deepen the investigation on the case presented in this paper, the SigniFYIng APIs component. This tool provides artifacts and procedures that support an in-depth reflection about the communicability of APIs.

Finally, we intend to further explore concepts from Collaborative Systems (discourse awareness and coordination awareness) to assess if they could help us by framing the communicability breakdowns in a collaborative setting.

REFERENCES

[1] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," ACM SIGCOMM Computer Communication Review, vol. 39, no. 1, p. 50, Dec. 2008.

[2] M. Kavis, Architecting the cloud: design decisions for cloud computing service models (SaaS, PaaS, and IaaS). Hoboken, New Jersey: Wiley, 2014.

[3] R. O. Prates, C. de Souza, and P. Assis, "Categorizing communicability evaluation breakdowns in groupware applications," CHI-SA 2001, 2001.

[4] G. Lawton, "Developing Software Online With Platform-as-a-Service Technology," Computer, vol. 41, no. 6, pp. 13–15, Jun. 2008.

[5] J. P. D. Preti and L. V. L. Filgueiras, "Interação em nuvens," in Proceedings of the IX Symposium on Human Factors in Computing Systems, 2010, pp. 209–212.

[6] S. Kolb, "Making Platform as a Service offerings comparable – Ecosystem profiles for portability matching." [Online]. Available: https://PaaSfinder.org. [Accessed: 08-Aug-2016].

[7] C. S. de Souza and C. F. Leitão, "Semiotic Engineering Methods for Scientific Research in HCI," Synthesis Lectures on Human-Centered Informatics, vol. 2, no. 1, pp. 1–122, Jan. 2009.

[8] C. S. de Souza, R. de G. Cerqueira, L. M. Afonso, R. R. de M. Brandão, and J. S. J. Ferreira, Software Developers as Users. Cham: Springer International Publishing, 2016.

[9] P. Mell and T. Grance, "The NIST definition of cloud computing," 2011.

[10] C. S. de Souza, The semiotic engineering of human-computer interaction. Cambridge, Mass: MIT Press, 2005.

[11] C. S. de Souza, C. F. Leitão, R. O. Prates, and E. J. da Silva, "The semiotic inspection method," 2006, p. 148.

[12] W. Y. Chang, H. Abu-Amara, and J. F. Sanford, Transforming Enterprise Cloud Services. Dordrecht: Springer Netherlands, 2010.

[13] S. Roy, B. Chakraborti, P. K. Pattnaik, and R. Mall, "Usability evaluation of some popular PaaS providers in cloud computing environment," Journal of Theoretical and Applied Information Technology, vol. 80, no. 2, p. 315, 2015.