Extensible User-friendly Rule System for connecting Internet Services

André Filipe Pereira Rodrigues

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Advisor: Dr. Miguel Filipe Leitão Pardal
Co-Advisor: Dr. José Manuel da Costa Alves Marques

Examination Committee

Chairperson: Dr. Daniel Jorge Viegas Gonçalves
Advisor: Dr. Miguel Filipe Leitão Pardal
Members of the Committee: Dr. Alberto Manuel Rodrigues da Silva

May 2016

Acknowledgments

Without the support of several people, this work would not exist. I want to dedicate this section to them.

First, I want to thank my advisor Miguel Pardal for the excellent advisoring and all the technical and emotional support. I also want to thank my co-advisor José Alves Marques for his critical feedback. On a more personal level, I want to thank my girlfriend Patrícia Amorim for supporting me during all this time and for discussing ideas that led to a better solution and document. I also want to thank my family and close friends for all the support. Finally, I want to thank all users (both developers and end-users) for spending their precious time helping me validate this work.

Abstract

The number of services available in the Internet increases daily. This number is due to increase even more when the Internet of Things arrives and more services are needed to monitor and control everyday objects that will be connected to the network. Services provide information about their environment and functionality that can be modeled as events and as actions, respectively. These events and actions can be used in rules to automate specific tasks: when the events occur, the actions are executed. Most existing applications need developers (or, at least, power users) to create the rules. There are applications that allow end-users to create rules, but only simple ones. This dissertation presents a solution that allows end-users to create rules that connect Internet services that are more expressive, yet still simple enough. Developers are still necessary to add support for new services. The solution was evaluated with end-users and developers, showing that it is possible to create an extensible rule platform that allows Internet services to work for common users, improving some of their daily tasks.

Keywords: End-User Programming, Trigger-Action Programming, Event Processing, Task Automation, Internet Services

Resumo

O número de serviços disponíveis na Internet aumenta diariamente. Este número deve aumentar ainda mais quando a Internet das Coisas chegar e mais serviços forem precisos para monitorizar e controlar objectos do dia à dia que estarão ligados à rede. Os serviços fornecem informação sobre o seu ambiente e funcionalidades que podem ser modeladas como eventos e como acções, respectivamente. Estes eventos e acções podem ser utilizados em regras para automatizar determinadas tarefas: quando os eventos ocorrem, as acções são executadas. A maioria das aplicações existentes requerem programadores (ou pelo menos utilizadores avançados) para criarem as regras. Existem aplicações que permitem a utilizadores principiantes criarem regras, mas apenas regras simples. Esta dissertação apresenta uma solução que permite a utilizadores principiantes criarem regras que ligam serviços existentes na Internet que são mais expressivas mas ainda suficientemente simples. Os programadores ainda são necessários para adicionarem suporte para novos serviços. Este sistema foi avaliado com utilizadores principiantes e programadores, mostrando que é possível criar uma plataforma de regras extensível que permite aos serviços disponíveis na Internet trabalharem para os utilizadores comuns, melhorando algumas das suas tarefas diárias.

Palavras-chave: Programação por Utilizadores Principiantes, Programação Despoletar-Acção, Processamento de Eventos, Automatização de Tarefas, Serviços disponíveis na Internet

Contents

List of Tables

List of Figures

Acronyms

API Application Programming Interface. 5, 9, 12, 16, 17, 23, 27

BPA Business Process Automation. 1

DAG Directed Acyclic Graph. 8, 9

EC Event Consumer. 2, 3, 7, 8, 12–19, 22, 23, 27, 28

EP Event Provider. 2, 3, 7, 8, 12–19, 22, 23, 28

EPA Event Processing Agent. 2, 3, 7–9, 13–17, 19, 22, 23, 27–31

EPE Event Processing Engine. 2, 7, 8, 15

EPN Event Processing Network. 2, 7–9

HTTP Hypertext Transfer Protocol. 16

IFTTT If This Then That. 1–3, 9, 10

IST Instituto Superior Técnico. 28

IT Information Technology. 27

JSON JavaScript Object Notation. 16

REST Representational State Transfer. 16, 17

RSS Rich Site Summary. 23

SaaS Software as a Service. 34

URL Uniform Resource Locator. 7, 14, 23

XML eXtensible Markup Language. 8

YAML YAML Ain't Markup Language. 15

Chapter 1

Introduction

Nowadays end-users perform several tasks using services available in the Internet. Here is an example of how people use Internet services: Andreia is a student that is finishing her Master Thesis in Computer Science. Meanwhile, she is already searching for job offers from interesting companies. For this reason she joined Landing Jobs[1], a tech-hiring marketplace. She is looking for jobs that require a Java[2] developer but does not want to waste her time every day searching. This example illustrates a possible task - finding job offers - that is currently performed by an end-user but could be automated. For example, she could receive an email everytime a new Java job offer appeared in Landing Jobs.

This is an example of the type of tasks that have been addressed before by Business Process Automation (BPA). BPA is the use of information technology for the automation of activities or services that accomplish a specific function or workflow [1]. The vast majority of automations are performed by a developer, or at least by a power user – a knowledgeable and sophisticated user of computers. However, end-users should also be able to automate their tasks. To allow end-users to automate tasks, one must first understand how can tasks be automated.

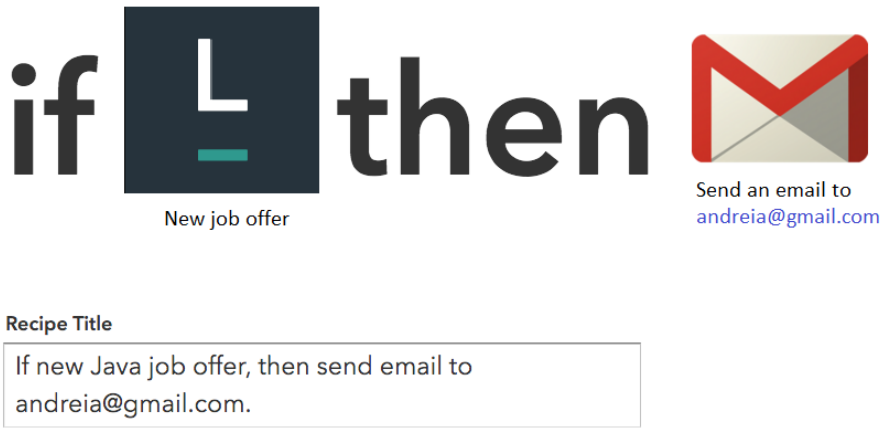


Figure 1.1: Illustration of a rule in IFTTT.

There are applications that already allow end-users to automate tasks, for instance, If This Then That (IFTTT) [2]. IFTTT is a proprietary rule platform that allows end-users to create simple event processing rules that use Internet services. Figure 1.1 illustrates a rule in IFTTT that automates the example task described above. The rule works as follows: "if a new Java job offer appears, then send an email to

[1] Landing Jobs - https://landing.jobs. Last accessed on 9 May 2016.
[2] Java programming language - https://www.java.com/en/. Last accessed on 9 May 2016.

andreia@gmail.com". IFTTT provides an easy-to-use interface, designed for users that do not know how to write code, where users create rules with the form "If event Then action". The events and actions are provided by the Internet services that are supported by the platform.

One characteristic of IFTTT is that its rules are very simple. A consequence of that simplicity is that Andreia's mailbox is always full. That is why she wants to create more advanced rules. One of the rules she would like to have is to receive weekly emails with the new Java job offers instead of an email per job offer. Another rule she would like to have is to receive an email describing whether the demand for Java developers is increasing or decreasing, by comparing the number of offers of the current week with the average of the previous 8 weeks. This rule would allow her to gain more insight about the job market conditions. These rules are not supported by IFTTT, and the existing applications that support this type of rules are designed for developers. This is one of the challenges of this work: how to create an application that allows end-users to automate tasks by creating rules that are more expressive, yet still simple. To answer this question, we need to understand how these applications work.

As seen in IFTTT, one common approach used to automate tasks is automating with events. An event is an abstraction of something that has happened [3]. Events can be related to real-world occurrences or to virtual domains, like computer games. The main reason to learn that something has happened is that it creates the opportunity to act upon it. Event processing is the field that studies the operations applied to events [4]. Event Processing Engines (EPEs) allow users to create Event Processing Networks (EPNs), where Event Providers (EPs), Event Processing Agents (EPAs) and Event Consumers (ECs) are the fundamental building blocks [5]. The events flow through the EPN, starting at the EPs, then are transformed by the EPAs, and in the end are consumed by the ECs. For example, an EPN could have a job offer EP, an EPA that filters job offers by Java, and an EC that sends emails.

The EPNs use EPs and ECs to detect and act upon events, respectively. Nowadays users depend on Internet services to perform their tasks more efficiently and effectively. These services provide both data and functionality that can be used in task automation, such as as EPs and ECs. In addition, the number of services available in the Internet increases daily, handling more and more use cases.[3] The Internet of Things [6] contributes to this tendency as it is needed to monitor and control everyday objects that are connected to the Internet. In the example above, the rules depend on two Internet services: Landing Jobs and an email provider. Landing Jobs provides job offer events and the email provider allows to send emails. Gmail[4] could be used as an email provider, as it allows users to know when a new email arrived (EP) and to send emails (EC). These facts strengthen the view that an application that allows end-users to automate tasks should support Internet services. The application should support as many Internet services as possible, to allow the automation of as many tasks as possible.

The EPAs are used in EPNs to transform low-level events into higher-level events that contain more information than the ones before the transformation. There are three fundamental types of EPAs [4] :

- **Filtering** - Events are filtered by attribute values;
- **Transformation** - The creation of events from other events, either by composing events, by enriching the event with more information, etc;
- **Pattern Matching** - Events are matched against a pattern, normally associated with a context, such as temporal context, to reason about what happened during a time frame.

[3] Rise of Internet services - http://techcrunch.com/startups/. Last accessed on 9 May 2016.
[4] Gmail - https://mail.google.com. Last accessed on 9 May 2016.

Each Andreia's rule uses at least one type of EPA. The first rule uses the filtering type because it filters job offers depending on whether they contain the tag "Java". The second rule uses the transformation type as it groups all job offer events that happened during the week in a new event. The third rule uses pattern matching because the result (demand increased or decreased) depends on comparing the number of job offers of the week with a pattern (the average number of job offers of the previous 8 weeks).

The challenge with existing solutions [2, 7, 8], like the aforementioned IFTTT, is that they only support the filtering EPA. This limits the number of tasks that can be automated by end-users and the expressiveness of the rules. Rules that use more advanced EPAs are more expressive. Other solutions [9] that support more EPAs are designed for developers. With so much information available, which will increase over time, making event processing accessible to end-users is seen as a key issue in the event processing community.[5] Another limitation that arises in these applications is related to who can add support for new Internet services. Most applications that target end-users to automate tasks are commercial and only the application owners can add support for new Internet services. However, the higher the number of Internet services available in platform, the more use cases it handles. One possible solution to this limitation is to allow third-party developers to add support for new Internet services by integrating them in the rule platform. Figure 1.2 illustrates the roles of *developers* and *end-users* in this rule platform. The developers create the code for the building blocks (EPs, EPAs, ECs) and the end-users use the building blocks in rules.

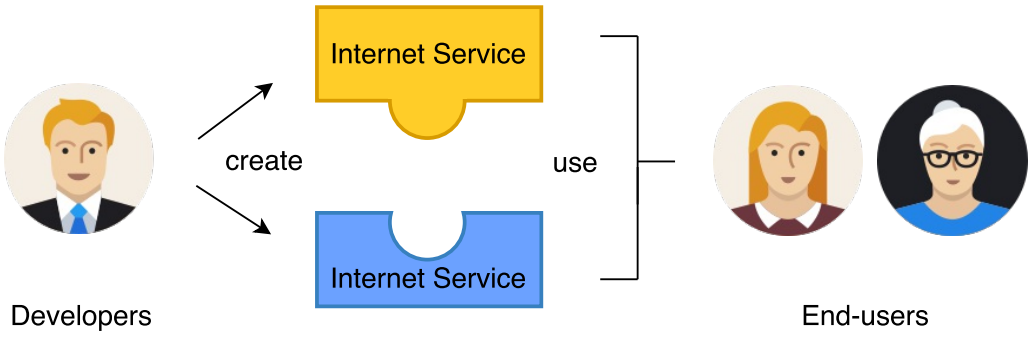


Figure 1.2: The roles of developers and end-users in the rule platform.

In summary, the challenge that this work tries to address is how to create a platform that, on one side, allows *end-users* to automate tasks by creating expressive event processing rules that depend on Internet services, and, on the other side, also allows *developers* to extend the platform with new functionality by integrating new Internet services.

1.1 Contribution

The contribution of this dissertation is the architecture design for a platform that allows *end-users* to automate tasks by creating expressive, yet still simple event processing rules that depend on Internet services and; *developers* to extend the platform by integrating new Internet services. An open-source implementation of this architecture was implemented, that can be used by end-users and extended by developers. An analysis of the results obtained from the evaluation performed with developers and end-users is also given, along with a description of possible improvements to be implemented in the future.

[5]For example, see Opher Etzion's presentation - http://www.slideshare.net/opher.etzion/on-personalization-of-eventbased-systems. Last accessed on 9 May 2016.

1.2 Dissertation Outline

The rest of this document is organized as follows. Chapter 2 provides a background on Internet services and event processing systems, and discusses how other systems try to solve the problems described above. Chapter 3 presents a solution for these problems and describes the inner workings of a prototype that implements this solution. Chapter 4 describes how the prototype was evaluated with both developers and end-users and summarizes the results obtained. Finally, Chapter 5 concludes the document, summarizes the contributions of this work and proposes future work to improve this system.

Chapter 2

Background

This chapter starts by presenting a model for Internet services and explains why they are useful for automating tasks. Then it provides a thorough explanation about event processing and compares several event processing engines. Afterwards, it explains how existing systems allow end-users to automate tasks using Internet services. Lastly, the chapter is concluded with a summary of the lessons learned with the existing state-of-the-art and a description of what should be improved.

2.1 Internet Services

An Internet service is a software application, available in the Internet, that provides some service. Typically, Internet services provide two interfaces: one is the user interface, which allows humans to use the software; the other interface is an Application Programming Interface (API) that allows other software applications to communicate with the service. One problem that arises in applications that use other applications to obtain information or to perform some procedure is that these resources might not available to all users. Some information can only be accessed by some users. For this reason, an application that uses Internet services should allow users to access their information of a given service by using an authentication protocol that identifies them in the respective Internet service. As the data accessed by the application belongs to the users themselves, there is no need for other access control mechanisms and the users are free to use it as they want.

We performed a survey to understand the type of information and functionality available in Internet services, and how can other applications access user information available in Internet services. Table 2.1 contains the results of the survey for the most relevant Internet services. We reached two conclusions. The first is that there is an opportunity for users to automate tasks by using information available in Internet services, as they provide relevant information (e.g. new job offer; new email; new message; new item found by the search on Ebay; new profile view) and functionality (e.g. send email; send message; accept friend request). The second is that most Internet services use OAuth [10], an open and secure authorization protocol that allows users to authorize applications to access their information available in an Internet service (e.g. allow LinkedIn to access some information available in Facebook). Therefore, it is possible to build a system that uses information and functionality available in Internet services, with secure use of the users' personal data.

Service	Personal Data	Events/Notifications	What users do with them?
Ebay	Location, Credit card, Products	New item found by this search; Bought item; Auction ended; New auction proposal; Received feedback.	View item price/reviews; View track progress; Give feedback to seller; View auction progress.
Facebook	Email, Photos, Contacts, Location, and others	New message; New tagged image/post; New post in event/group; Post scheduled to page submitted; New event; New comment.	View/Respond message; Like/Comment image/post; View event; View post; View comment.
Fitbit	Calories, Number of steps, Distance, Heart Rate, Location, Fitness goals, Food logs, Sleep logs	Received a friend request; Completed/Missed goal.	Accept/Reject friend request; View user profile; Share goal; View statistics about that goal.
Landing Jobs	Email, Curriculum	New job offer; Received recommendation; Sent recommendation; Submitted application for job offer.	View job offer; View recommendation; Submit application for job offer.
LinkedIn	Email, Photos, Curriculum, Contacts, Friends, Phone number, Location	Someone has viewed your profile; New message; New connection request; New post in group; New comment; Recommended in something (e.g. Java)	View/Respond message; View user profile; Accept/Reject connection request; View/Respond job offer; View/Create/Comment post; View user profile.
Outlook Calendar	Events	Another user added/ removed an event; Birthday today; Event reminder;	View event; Close reminder; Send an email to the birthday person.
Outlook Mail	Emails	New email arrived.	Create rules such that, when the email arrives, it is: moved to a specific folder, forwarded to another email and deleted; Send an email.

Table 2.1: Survey about data and functionality available in Internet services.

2.2 OAuth

Before we can access Internet services and their (sensitive) personal data, we need adequate permissions. OAuth is a protocol that allows to obtain these permissions. This protocol must be supported in an application that uses information and functionality available in Internet services because most Internet services require it. This section briefly explains how OAuth works.

The first step before using the OAuth protocol is to register the application in the Internet service to be accessed. Internet services provide a form where developers provide details about their applications.

This step is required because one of the details specified when registering their applications is the callback url, which is an Uniform Resource Locator (URL) that is invoked when a user authorizes the application to access his information. Once the application is registered, two credentials (client identifier and client secret) are provided, which are used by the Internet service to identify the application when processing its requests. With these credentials, the application can start using the OAuth protocol.

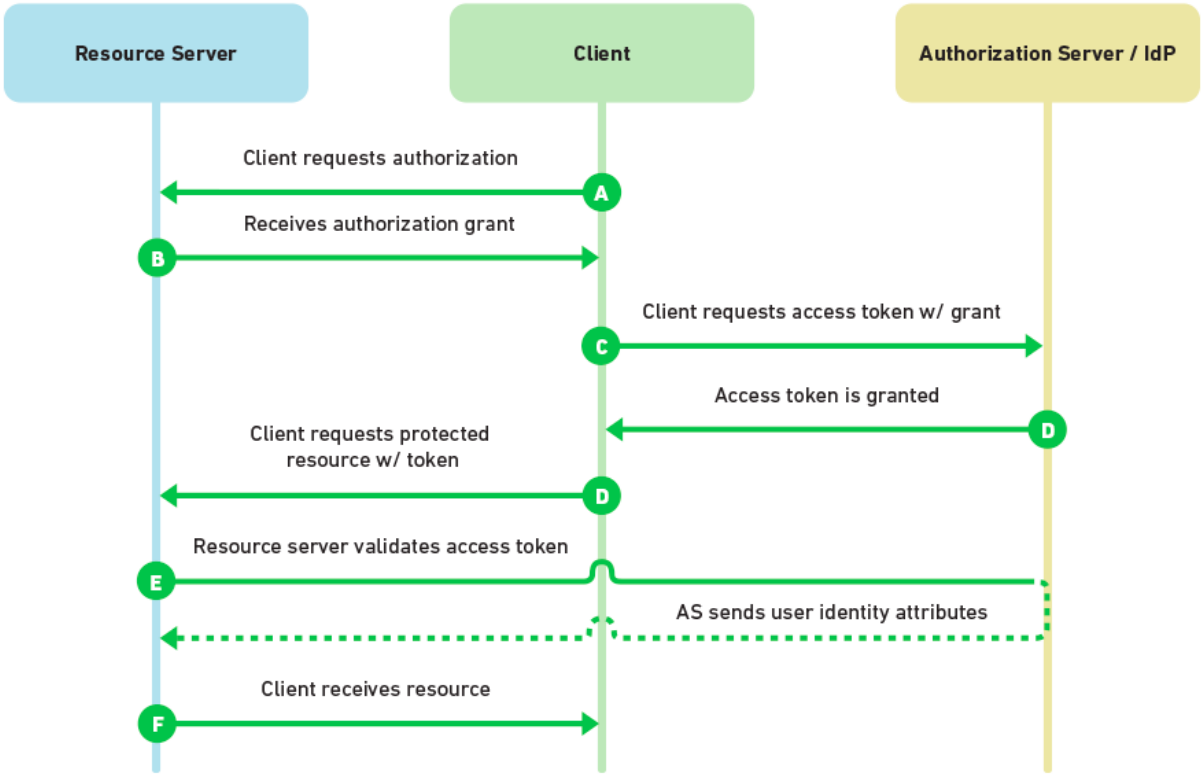


Figure 2.1: Abstract OAuth authorization flow.

An abstract flow of the OAuth protocol is displayed in Figure 2.1. It is used when an application requests access to a protected resource of an Internet service. It is abstract because there are subtle variations depending on whether a browser or a mobile application is used. The flow starts with the client application requesting authorization to access a resource to the resource server (step A). If the resource owner (the user) authorizes, the resource server returns an authorization grant (step B). The authorization grant is then used to request an access token to the authorization server (step C). The authorization server returns an access token, which is used to access protected resources (step D). Every time the client application requests a protected resource using the access token, the resource server sends the access token to the authorization server for validation purposes (step E). If the access token is valid and the client application has permission to access the requested resource, the resource server returns it (step F).

2.3 Event Processing Engines

Event Processing Engines (EPEs) allow users to specify Event Processing Networks (EPNs), which are composed by Event Providers (EPs), Event Processing Agents (EPAs) and Event Consumers (ECs). Figure 2.2 illustrates an EPN composed by one EP, two EPAs and two ECs. The events start at the EP, which are then sent to the EPAs that transform them, and finally are consumed by the ECs.

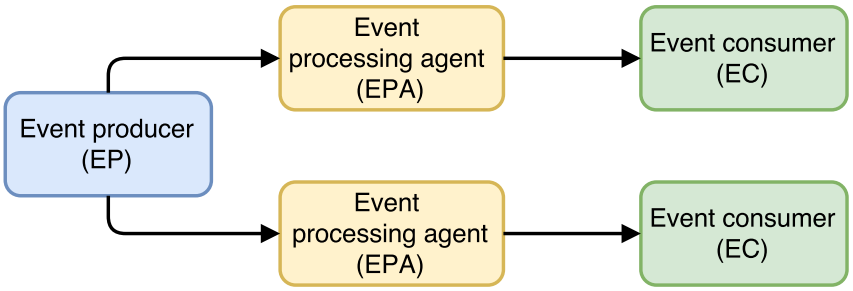


Figure 2.2: An event processing network composed by one event provider, two event processing agents and two event consumers. Adapted from [4].

There are several aspects to take into account when comparing EPEs [3]. One aspect is the representation of the EPN. This aspect determines the type of users that can use the system. There are EPEs that require users to specify the logic using programming languages [11, 12, 13, 14, 15]. Other EPEs use an eXtensible Markup Language (XML) vocabulary to specify the rules [16, 17, 18]. In these EPEs, the EPN is created by developers. Other EPEs use domain specific languages (DSL) that look similar to human language [19, 20, 21]. In these EPEs, the EPN is created by power users. To facilitate usage by end-users, new systems started using other visual approaches. For example, SARI [22] uses a Lego-like visual approach, where end-users assemble rules using solution templates that vary on the domain. Other systems allow users to create EPNs with the form "IF event THEN action", along with an intuitive user interface [2, 7].

Another aspect is concerned with the format of the EPN. This aspect determines the level of expressiveness of the EPN and the type of users that can use the system. An EPN that allows more EPs, more EPAs and more ECs is more expressive but is also more difficult to create and understand. The existing systems used by developers or power users have the form of a Directed Acyclic Graph (DAG) [23]. The existing systems used by end-users have the form "IF event THEN action".

A third aspect to be considered when comparing EPEs is whether it runs locally or is distributed. This aspect influences the availability and the performance of the system. Some EPEs, such as Apache Flink [11], Apache Samza [12], Apache Spark [13], Apache S4 [14] and Apache Storm [15] are distributed in order to scale with the amount of events and rules and to provide fault-tolerance. In case the system is distributed, it has to consider how to process events, how the memory is managed, how the data is shared between nodes, which message guarantees should be used, how the data is serialized, and others. The message guarantees vary between at-least-once, where a message can be sent one or more times; at-most-once, where a message will only be sent one time; and exactly-once, where a message will always be processed one time.

A fourth aspect that differentiates EPEs is the processing model. The choice of the processing model influences the performance and the behavior of the EPNs. For instance, Apache Spark [13] uses batch processing, which means that it stores a lot of data before processing it. This approach is better in the case where there is a lot of data to be processed. On the other side, Apache Storm [15] uses stream processing, which means that events are processed as soon as possible. This approach is better in the case where sub-second latency is desired. There is also another processing model called micro-batching, which attempts to be in-between the other two approaches. For example, the Trident framework[1] of Apache Storm uses micro-batching to achieve exactly-once message guarantees.

[1] Trident framework - http://storm.apache.org/releases/1.0.0/Trident-tutorial.html. Last accessed on 9 May 2016.

2.4 Rule systems that connect Internet services

This section describes existing systems that allow users to automate tasks by creating event processing rules that depend on Internet services: Node-RED and IFTTT. This section refers to these systems because they make different trade-offs regarding the type of users they target and the type of rules they support.

Node-RED is a visual tool that allows developers to connect hardware devices and Internet services using node.js [9]. By connecting the different nodes, the developer is passing events between nodes and performing actions. Developers can also create NPM[2] (a package manager for node.js) packages that can be used by all applications, thereby extending the platform. The EPNs created in Node-RED have a structure of a DAG. This graph structure allows the creation of richer event-based rules because a node can receive events from several nodes and also send events to several nodes. In contrast, it is harder for end-users to understand it. Figure 2.3 shows an example of a rule in Node-RED. The rule starts with a payment node that emits payment events. These events pass through the payment network node that is responsible for deciding which payment processor should be used. The payment processor node that receives the event invokes the Internet service API.

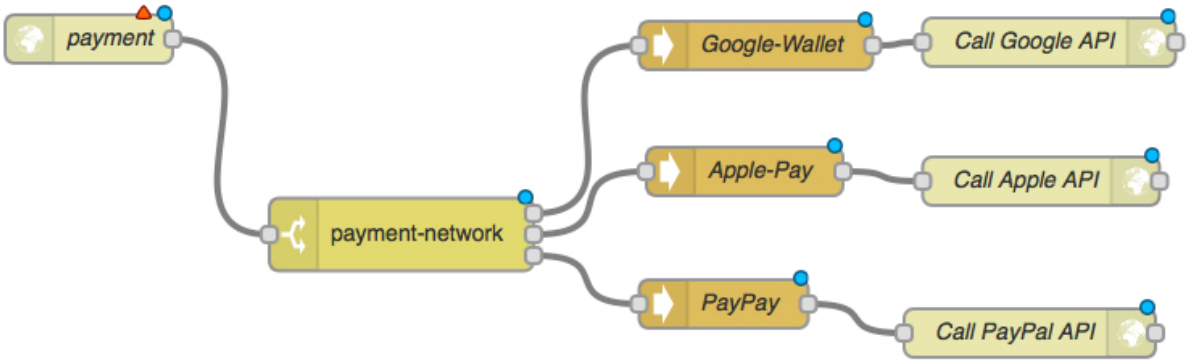


Figure 2.3: A rule in Node-RED.

If This Then That (IFTTT) is a proprietary platform that enables end-users to connect events to actions, as in "If event Then action" [2]. Its user interface is simple and easy to use. Channels provide events and actions for a specific topic or matter (e.g. Android Channel). A channel can depend on a device (e.g. Android smartphone) or on an Internet service (e.g. Dropbox[3]). To use a channel that depends on a device, the user must install the IFTTT application in that device. To use a channel that depends on an Internet service, the user has to authorize IFTTT to use the Internet service on his behalf. Figure 2.4 illustrates a rule that uses the Android Location and the Dropbox channels. This rule saves the user's location in a text file stored in Dropbox. This platform allows end-users to create simple event processing rules without requiring them to understand the underlying event processing infrastructure, with the limitation that the only type of EPA supported is the filtering type. Another limitation in IFTTT is that it does not allow developers to add support for new Internet services or devices, or even extend the existing ones with new events and actions. In this sense, it is a closed platform.

[2]Node Package Management - https://www.npmjs.com/. Last accessed on 9 May 2016.
[3]Dropbox - https://www.dropbox.com. Last accessed on 9 May 2016.

Figure 2.4: A rule in IFTTT that uses the Android Location and the Dropbox channels.

2.5 Summary

From the survey of Internet services, we learned that there is an opportunity to build a system that allows end-users to automate tasks that depend on Internet services. Internet services allow users to take advantage of their personal data to automate tasks. For applications to access users' data, they must first ask them for permission. OAuth is the most used protocol that allows applications to ask users for permission to access their data.

Regarding event processing systems, we learned that the expressiveness of event processing rules is tied to their format. On one hand, systems like IFTTT allow end-users to create simple event processing rules due to its "If then" rules. On the other hand, Node-RED allows developers to create expressive event processing rules due to its graph structure.

Chapter 3

Solution

We are now describing a solution we called shAPPerd[1], that improves the existing ones in two ways: First, the system is extensible, in that it allows third-party developers to add support for new Internet services and to add new functionality to existing ones. Second, it allows end-users to create expressive event processing rules to automate tasks that depend on Internet services. From the system's perspective, the requirements of this solution is that it should be able to execute rules in an efficient manner and it should be extensible (it should be possible to add support for new Internet services). From the user interface's perspective, the requirements of this solution is that it should allow end-users to create rules in a friendly manner, much like the Andreia's example described in Chapter 1. These requirements were validated with the evaluation described in Chapter 4.

3.1 Architecture Overview

The solution is divided in three parts: Internet Services, Backend and Mobile Frontend. It is represented in Figure 3.1. The Internet services contain the information and the functionality that will be used in the event-processing rules. The backend is the processor of the information. It is responsible for communicating with the Internet services (to get information and to use available functionality) and the mobile frontend (to receive requests and send information). Additionally, it is the "place" where all information is stored. Furthermore, it manages the rules created by the users and makes them work as expected. The mobile frontend is the interface by which the end-users interact with the solution. We decided on a mobile application because it may be useful in the future to implement new features (such as accessing information or using functionality available in the mobile device).

A prototype was created that implemented this architecture. It is open-source and available in Github[2], which is an online repository hosting service. The programming language used in the implementation is Java 8[3]. To build and execute the prototype, Maven 3[4] was used. The details of the implementation are explained in the rest of this chapter.

[1] shAPPerd is the union of the terms application and shepperd, and symbolizes a system that keeps the apps working better together.

[2] shAPPerd Implementation in Github - https://github.com/furypt/shAPPerd. Last accessed on 9 May 2016.

[3] Java programming language - https://www.java.com/en/. Last accessed on 9 May 2016.

[4] Maven - https://maven.apache.org/. Last accessed on 9 May 2016.

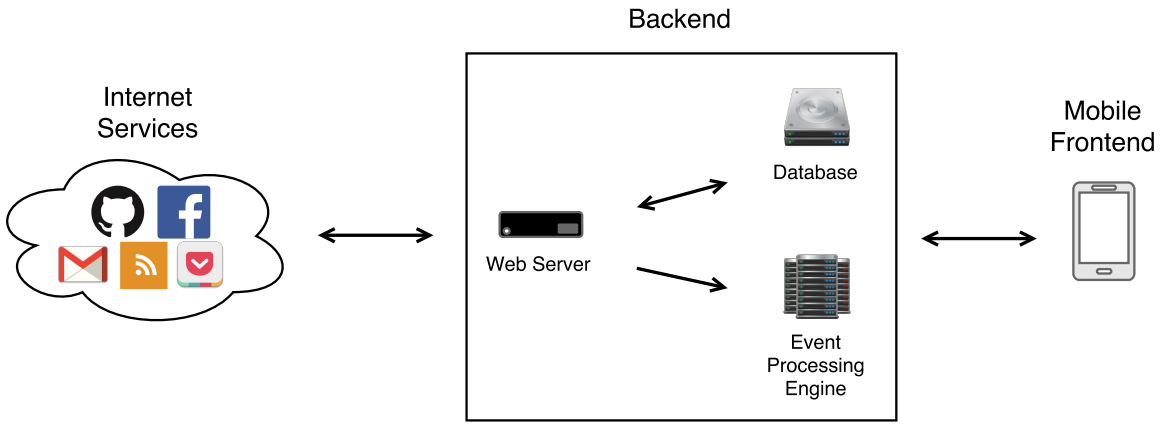


Figure 3.1: The three parts that compose the solution: Internet Services, Backend and Mobile Frontend.

3.2 Internet Services

To allow the creation of rules that depend on user information available in Internet services, the prototype supports the OAuth protocol. Instead of implementing this protocol, an already existing open-source library was used, called Google OAuth Java Client[5]. The prototype allows developers to write code that uses the Internet services' public API to implement new EPs and ECs. When a service requires a user account, the OAuth protocol is used.

3.3 Backend

As mentioned before, the backend component illustrated in Figure 3.1 is responsible for storing the information, communicating with both the Internet services and the mobile frontend and to make the rules work as expected. To handle these tasks, the backend is divided in three main components: Event Processing Engine, Database and Web Server. These components depend on several building blocks, which are explained first.

3.3.1 Building Blocks

Channel

This component represents an Internet service. It is used to group related EPs and ECs. It is also the entry-point for developers to extend the platform. They can create new channels, thereby extending the platform with new EPs and ECs. Figure 3.2 shows the relationships between the channels and the other building blocks. In the prototype, the channel contains the following information: name, description, event providers, event consumers and service dependencies. The name and description are used by end-users to understand which Internet service this channel represents. The event providers and the event consumers are the capabilities the channel provides. The service dependencies represent the authorizations that must be given by the user before using any of the capabilities of the channel. Currently, the source code must be updated whenever developers create a new channel.

[5]Google OAuth Java Client - https://github.com/google/google-oauth-java-client. Last accessed on 9 May 2016.

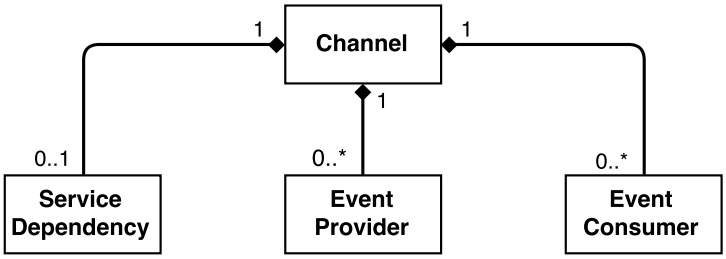


Figure 3.2: The channel building block.

Service Dependency

If a channel has a service dependency, it means that the user must authorize shAPPerd to access the Internet service on his behalf before using any of its capabilities (e.g. send email). A service dependency is fulfilled by using the OAuth protocol.

Rule

This component is used to automate a specific task by using EPs, EPAs and ECs.

As previously discussed, there is a trade-off between simple rules (ex: **when** event **then** action) and expressive rules (ex: **when** this **and not** that **then** perform this **and then** perform that). Rules that are more expressive can perform more advanced tasks but are also more difficult to understand and create. Their complexity depends on the number of conditions that trigger an event, on the amount of parameters in the events, on the number of EPAs and on the number of ECs. As this solution targets end-users, a different rule representation is used. This representation consists of one EP, zero or more EPAs and one EC (e.g. **when** this **then** that **and then** that). It can be observed in Figure 3.3. It allows rules to be more expressive than the simple rules because they can contain EPAs. Additionally, it is still easier to understand the rules with this representation than to understand the rules with the second representation given above because these rules can only have one condition as opposed to multiple conditions.

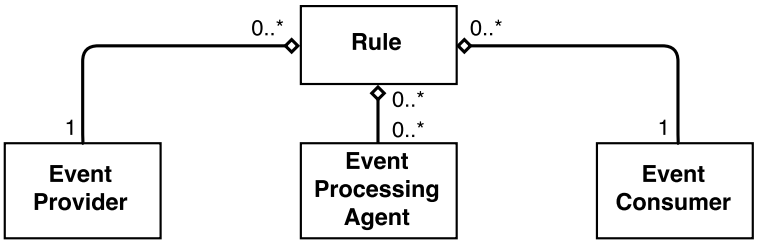


Figure 3.3: The rule representation used by the solution.

The rules' behavior can be observed in Figure 3.4. The events start at the EP, then are transformed by the EPAs, which may end up triggering the EC.

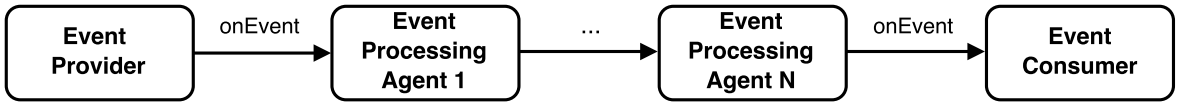


Figure 3.4: The behavior of a rule in the solution.

Event Provider

This is the component responsible for providing events. In the prototype, an EP contains the following information: name, description, parameters, variables and variable hints. The name and the description help the user understand how does this EP work. The parameters correspond to the information present in the events that this EP emits (e.g. a job offer could have as parameters the company name and the salary). Some EPs require additional information that must be specified by the user. The variables are used to ask that information to the user. The variable hints are used to help the end-user understand which information the EP is asking for.

There are two ways of detecting events: synchronously or asynchronously. Detecting events synchronously is achieved by polling an Internet service from time to time to see if something as happened. Detecting events asynchronously is achieved by receiving a notification from the Internet service announcing that something has happened. To handle both ways, there are two types of EPs: event listeners and event handlers. An event listener listens for events (synchronous). An event handler is triggered by webhooks[6] (asynchronous). A webhook is a defined URL that will be invoked when something has happened.

Event

The event is the component that is emitted by EPs when something has happened. It contains the information related to the situation that has occurred. In the prototype, the event is a list of key value pairs, where the keys are the parameters described by the EP and the values are the information about the situation.

Event Processing Agent

This component is used to transform events. It is used to implement the event processing features referred in Chapter 1. In terms of information, this component contains the same information as the EP: name, description, parameters, variables and variable hints. The difference is that this description explains how the EPA works. Consider, for example, an EPA that filters events by a certain parameter and a certain value. Its description could be **"Filters a parameter by a specific value"**. Its variables would be **"parameter"** and **"value"**. Possible variable hints could be **"age (in years)"** and **"10"**, respectively.

Event Consumer

This component is responsible for performing a procedure. The EC provides almost the same information as the EP: name, description, variables and variable hints.

[6]Webhook - https://sendgrid.com/blog/whats-webhook/. Last accessed on 9 May 2016.

One important detail about ECs is how they show the information of the event that triggers them. One approach is to allow the user to refer to the parameters of the event when providing the values for the variables of the EC. This approach requires a special syntax to know when the user is referring to a parameter of the event. It also requires the user to know about the parameters of the events emitted by the EP and the EPAs. For example, imagine a rule that uses a job offer EP that has the parameters company name and salary, and a send email EC that has a variable called **body**. In this approach, the user could assign the variable body the following value: "New job offer from company ${company name} paying ${salary}".

Another approach is to ask the user to select which parameters he needs when he chooses the EP (e.g. I need the company name and the salary of the job offers), and have the ECs to automatically use those parameters (e.g. a send email EC would send emails whose body would only have the company name and the salary of the job offer). This second approach requires a protocol to convert the required parameters of the event to a format that is understandable for the end-users. The prototype uses the second approach because it is easier for the end-users to use. The protocol used to convert the values of the required event parameters to readable text is YAML Ain't Markup Language (YAML)[7], a human-friendly data serialization format. Using this protocol, the end-users can get the information they need from the events without using special syntax.

User

This component represents a real user. It is used to store user data such as: email, password, the Internet services' credentials, the rules he has created and their state (enabled or disabled).

3.3.2 Main components

Here are explained the three main components referred in Figure 3.1. They use the building blocks described above to implement the required functionality.

Event Processing Engine

This component is responsible for deploying the rules. Deploying a rule means deploying all its components (EP, EPAs and EC). The lifecycle phases of a deployment are: install, start, stop and uninstall. This component abstracts theses phases and provides an interface where each rule is either **on** or **off**.

The prototype uses Apache Storm [15] as the underlying EPE because it is very fast, has low latency and stable (it is used in several enterprise systems). In Apache Storm the rules are created using the Java programming language, the same programming language the rest of the prototype was developed with. The fact that Apache Storm is fast and provides exactly-once message guarantees is very important because that is what users are expecting. The fact that it has low latency is directly related to the type of rules that can be created in these systems. The lower the latency, the higher the number of tasks that can be automated. For example, a rule with an EC that is responsible for taking a photo might not be very useful if it takes several seconds or even minutes for the event to trigger the EC. Additionally, there are several resources available in the Internet that explain how to use Apache Storm, which turned out to be easy to use.

[7] YAML - http://www.yaml.org. Last accessed on 9 May 2016.

Database

This component is used for querying and persisting the information available in the platform, which includes:

- The channels that are available in shAPPerd, along with their EPs and ECs;
- The EPAs that are available;
- The users' information.

This prototype does not implement a database because persistency was not needed for the evaluation. Instead, the prototype stores information in memory using objects.

Web Server

This component is used for communicating with Internet services and the mobile frontend. In the case of the Internet services, it can receive Hypertext Transfer Protocol (HTTP) requests that correspond to events (information expected by event handlers). The web server also communicates with Internet services when it must fulfill service dependencies. In the case of the mobile frontend, the web server provides it the means to interact with the database (access and modify the information).

The prototype uses the Spark framework[8] to create a Representational State Transfer (REST) API that uses HTTP as the communication layer. This API is composed by fifteen endpoints to communicate with the mobile frontend (endpoints that represent user accounts, rules, channels, EPs, EPAs and ECs) and one endpoint to communicate with the Internet services. Table 3.1 contains the endpoints available in the API. Each endpoint used by the mobile frontend returns a JavaScript Object Notation (JSON) response with the requested information.

Endpoint	HTTP Verb	Description
/accounts	POST	Create account
/accounts/login	POST	Perform login
/rules	GET	Get created rules
/rules	POST	Create rule
/rules/:id	GET	Get rule with given id
/rules/:id	DELETE	Delete rule with given id
/rules/:id/enable	POST	Enable rule with given id
/rules/:id/disable	POST	Disable rule with given id
/channels	GET	Get channels
/channels/:name	GET	Get channel with given name
/event-providers/:channelName/:name	GET	Get event provider with the given name, that belongs to the channel with the given channel name
/event-processing-agents/	GET	Get event processing agents
/event-processing-agents/:name	GET	Get event processing agent with given name
/event-consumers/:channelName/:name	GET	Get event consumer with the given name, that belongs to the channel with the given channel name
/events/:id	POST	Send event to the event handler with given id

Table 3.1: Endpoints available in the shAPPerd API.

[8] Spark Framework - http://sparkjava.com Last accessed on 9 May 2016.

3.4 Mobile Frontend

This section explains the minimal functionality that should be available in the user interface to allow end-users to automate tasks and how the prototype implements this functionality. This component is also illustrated in Figure 3.1. The features that should be available in the user interface are:

Login:
The application must allow users to create an account and to perform login. This is necessary because some functionality (e.g. create rules; view installed rules) depends on the user.

View Channels:
It is important to allow users to view the channels available in the platform. This feature promotes explorability, as it allows users to see which Internet services are supported and what they allow them to do.

Create Rule:
The most important feature of the application is to allow users to create rules. This feature involves selecting the EP, the EPAs and the EC. This process should be as simple as possible.

View Rules:
Another feature that should be available is to allow users to view the rules they have created. Users should not have to memorize which rules they have previously created.

Enable Rule:
Users should have the means to enable or disable a rule whenever they want.

Delete Rule:
Users should also have the means to delete a previously created rule.

We created an Android[9] application that implements these features. The reason for developing an Android application was because we only had an Android smartphone. Another reason that motivated the choice for Android was that Android uses the Java programming language, which was already being used in the prototype. The Android application is divided in two parts: the REST API client that communicates with the shAPPerd backend and the user interface. The API client is a wrapper of the fifteen endpoints available in the backend. It is used to get and modify the information available in the backend. The rest of this section explains the user interface, which is how end-users interact with the application.

[9] Android - https://www.android.com. Last accessed on 9 May 2016.

The application starts with the login screen, that allows the user to create an account or to log in. Figure 3.5 shows the user interface of the login screen. The user has to specify an email and a password and, only afterwards, can create an account or perform login.

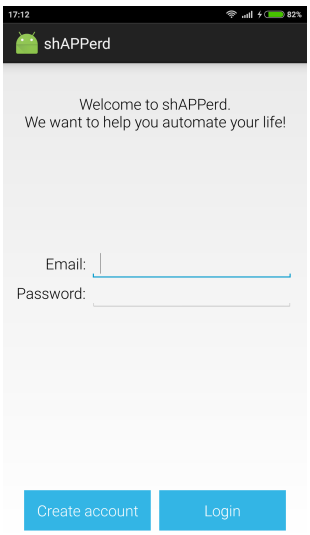


Figure 3.5: The login screen, where the user can create an account or log in.

After the login screen, the user jumps to the home screen. Here the user can either view the rules he has created or the channels available in the platform, as shown in Figure 3.6. In the left screen, the rules tab allows the user to view what rules he has already created and to create new ones. In the right screen, the channels tab allows the user to view the channels and correspondent EPs and ECs available in the platform. This way, the user can understand which Internet services are supported in shAPPerd and what information and functionality they provide. One important detail in the right screen is that there is no distinction between EPs and ECs. This is important because most users do not know what EPs and ECs are, and, in fact, they do not really need to know what they are. They just want to know what the system can do.

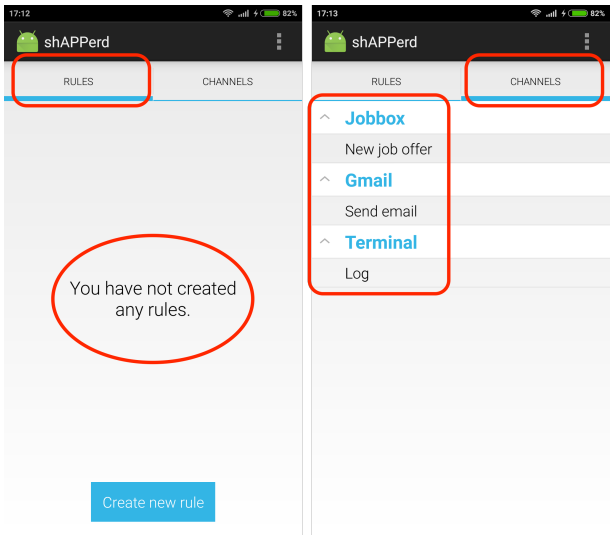


Figure 3.6: The home screen, where the user can view the created rules (left screen) or the channels available in the platform (right screen).

The process of creating a rule is shown in Figure 3.7. The rule illustrated in this figure is composed by one EP, one EPA and one EC.

The user starts in a screen where he can choose the EP by pressing the first **+** button. Note that the user does not have to know what are EPs and ECs. He just has to understand that, when what he selects in the first **+** button occurs, the thing in the place of the second **+** button will be executed. After pressing the first **+** button, the user can choose the EP. In this case there is only one channel that contains EPs. Channels that do not have any EPs are not shown. Subsequently, the user completes the selection of the EP by choosing the event parameters he needs. In this case the EP does not have any variables and, therefore, requires no additional information. Finally, the EP is selected and the application returns to the create rule screen, where the chosen EP is now highlighted.

By pressing the second **+** button, the user is able to choose either an EPA or an EC. To select an EPA, the user has to select the tab **transformations**. To select an EC, the user has to select the tab **actions**. The process of selecting and completing an EPA or an EC is the same as the EC.

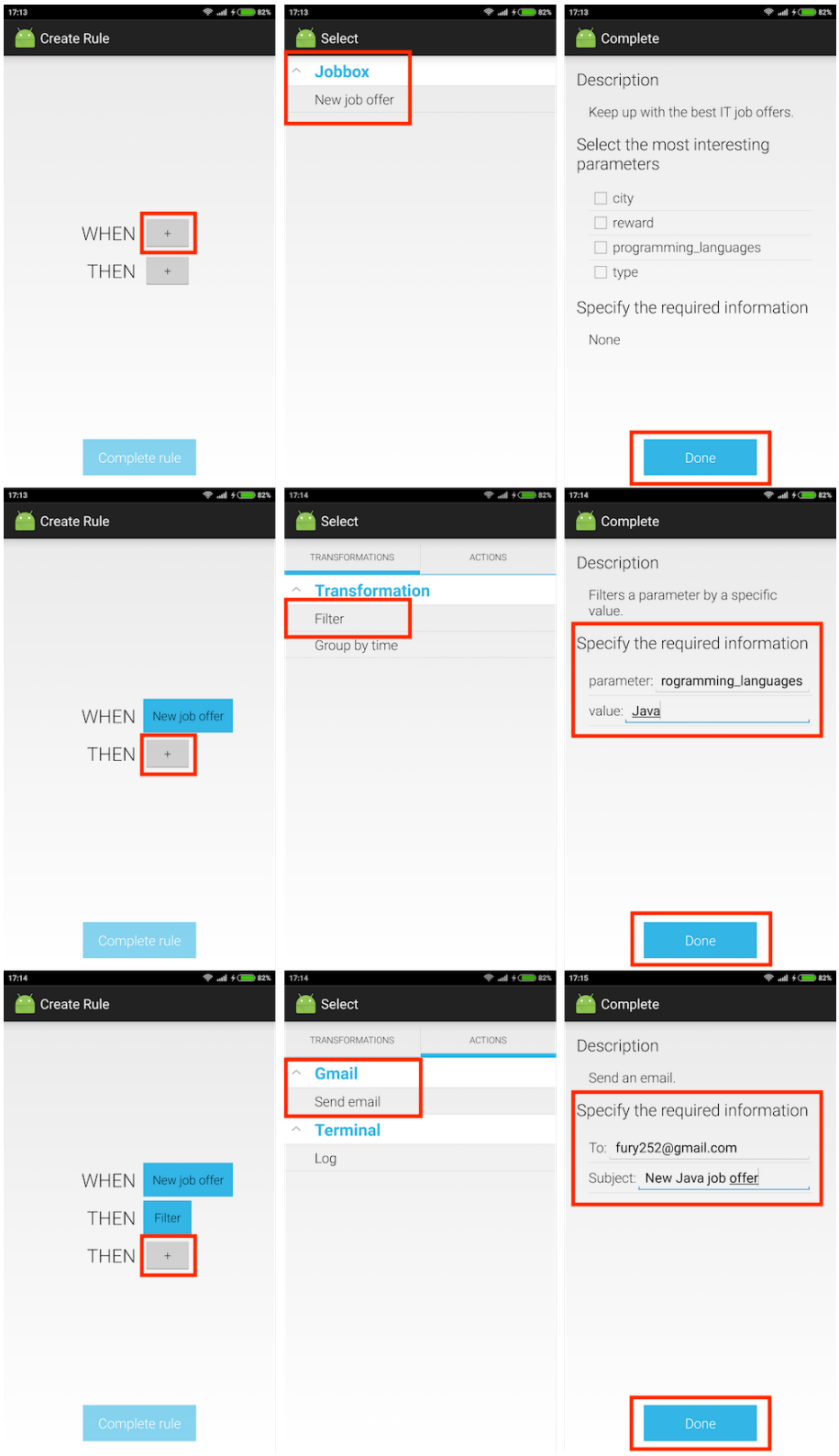


Figure 3.7: Steps taken to select the components of the rule (from left to right and from top to bottom).

After the selection process, the user must complete the rule to finally create it. The completion step can be observed in Figure 3.8. In the first screen, due to the usage of the **WHEN** and the **THEN** keywords, the final rule almost reads like plain english. This detail is important because it helps the end-user to understand the behavior of the rule. In the second screen, the user has to specify a title. This will help him later remember what it the role of this rule. After pressing the button in the end of

the screen, the user can now see the rule in the rules tab of the home screen (third screen). Now the user can enable the rule by pressing the button on the right. To disable it, he just has to press the same button again.

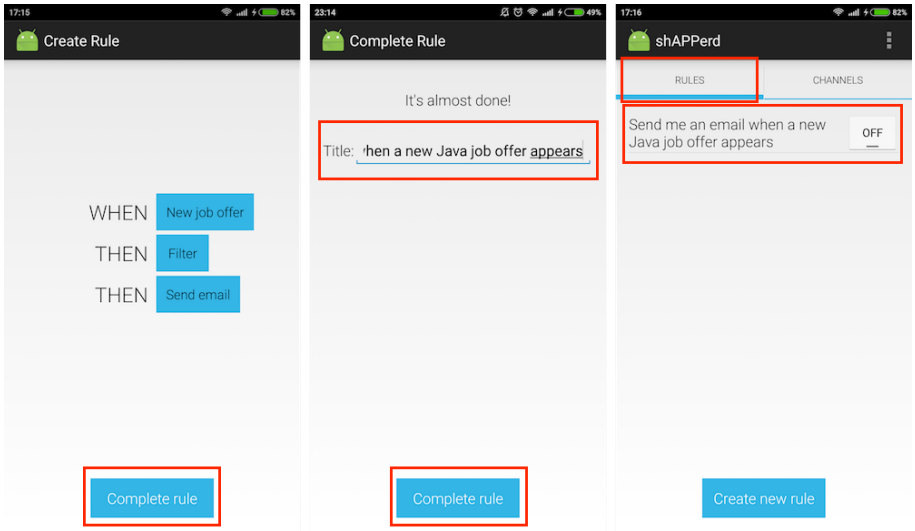


Figure 3.8: Steps taken to complete the rule (from left to right).

The user can also view more information about the rule by clicking on its tile. This leads to the screen presented in Figure 3.9. Here the user can remember the rule's logic and, if he wishes so, he can delete the rule by pressing the delete rule button.

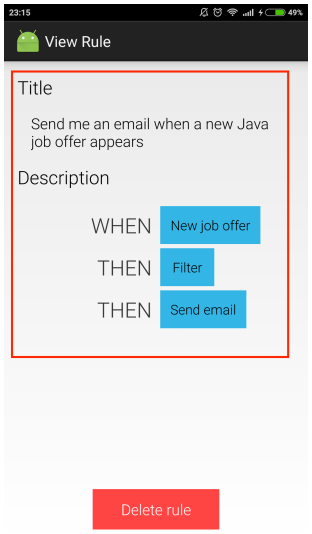


Figure 3.9: The screen that allows the user to view a previously created rule in more detail.

3.5 Summary

In this chapter was proposed our solution called shAPPerd. It allows end-users to create rules to automate tasks using Internet services. The chapter was divided in three sections: Internet Services, Backend and Mobile Frontend. Section 3.2 described how shAPPerd uses Internet services. Section 3.3 described the components that form the backend, from the building blocks to the main components,

and how they, when used together, implement the required functionality. It explained how developers can extend the platform with new EPs, EPAs and ECs. It also highlighted the importance of the rules' format and discussed two possible approaches to show the information available in the events, influencing the experience of the end-users. Finally, Section 3.4 referred which features should be available in the user interface and how the Android application implements these features. Every screen of the application was explained, highlighting all the details that make the application easier to use. With a working prototype, the next step is to evaluate it with both developers and end-users in order to understand how it achieves its design goals.

Chapter 4

Evaluation

This chapter explains the approach followed to verify whether our prototype solves the problems described in Chapter 1, namely, if end-users are able to create event-based rules and if developers can understand the API to create new EPs, EPAs and ECs. It is divided in two sections: developers and end-users. Each section contains the methodology, the setup, the results and discussion of the tests performed with the respective users. The methodology section explains how the solution was evaluated with the users. The setup section describes the conditions on which the users performed the evaluation. Finally, the results and discussion section highlights the most important results obtained from the evaluation with the users and discusses what they mean.

4.1 System evaluation (Developer's perspective)

This solution is only useful when there are several EPs and ECs, and these components are created by developers. To understand whether or not the solution is easily extended, we had to perform tests with developers.

4.1.1 Methodology

The developers were asked to solve two exercises, where the first consisted in implementing an EP and the second consisted in implementing an EC. The exercise descriptions are available in Appendix A. The EP to be implemented is an Rich Site Summary (RSS) feed reader[1] that, given an URL, polls the URL and emits as events the new items in the feed. The EC to be implemented, given a message and a filename, must append the message to the file in the local filesystem.

Before asking developers to solve the exercises, documentation was provided so that they could understand the system. The documentation for the solution is available in Appendix B. It answered as concisely as possible, the following three questions about the system: how to install it; how to use it; how to extend it. Developers are not able to use the system unless they know how to install it. If they do not know how to use the system, they cannot extend it. This is why it is important to provide documentation about how to install the system, explain what it is capable of and how to do it. Assuming that the developers know how to install and use it, they are then able to extend it. Finally, the documentation also taught developers on how they could extend the system. A questionnaire was also created, that guided the developers through the documentation and the exercises. It is available in Appendix C. It was

[1] RSS - http://www.whatisrss.com/. Last accessed on 9 May 2016.

used to learn about the developers' thoughts on the solution and whether they had already used similar systems.

4.1.2 Setup

The developers were asked to fill in the questionnaire and solve the exercises in an environment of their choice. The reason for not performing the tests with all the developers in the same environment was because the tests were performed during the holiday season. During this period there were not many people available to go to a place where the tests could be performed. Still, the developers were asked to perform the evaluation in a distraction-free environment and to do everything at once. They had 30 minutes to perform both exercises and then stop, even if they were not able to solve any of them.

4.1.3 Results and Discussion

The solution was evaluated with different developers during 3 sessions, with the difference that the materials (documentation and questionnaire) of each session are an improved version of the materials of the previous session. The first, second and third sessions involved 2, 5 and 18 developers respectively. The results of the most important questions available in the questionnaires are discussed below.

One question corresponded to the developers' opinion about the usefulness of the idea. The responses can be observed in Figure 4.1. From the figure, it can be concluded that most developers thought the idea was useful.

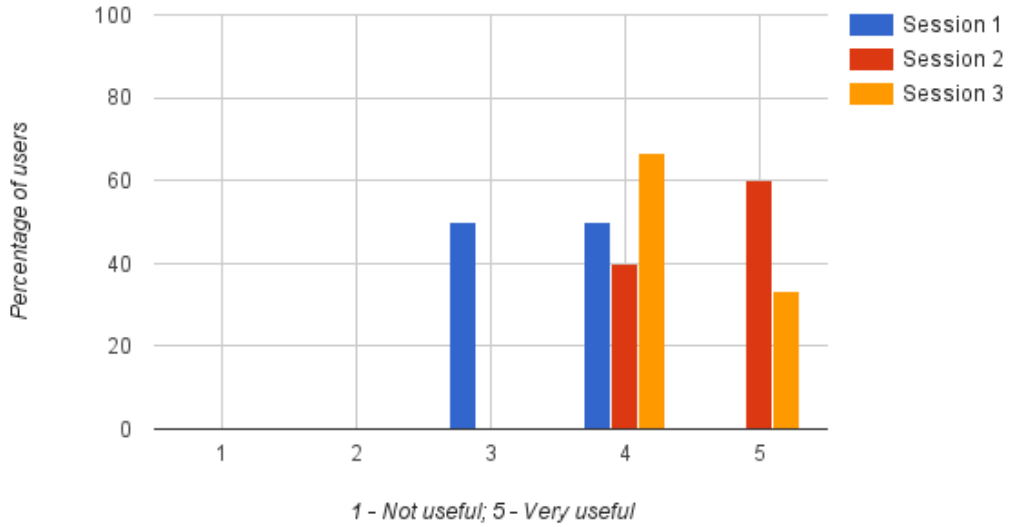


Figure 4.1: Developers' opinion about the usefulness of the idea.

Another question was asked to determine whether the developers had understood the system. The responses can be observed in Figure 4.2. The results show that, as the documentation improved, the developers' understanding of the system also improved.

Now follow three questions related to the exercises. The first question asked about the developers' opinion about the difficulty of the exercises. The responses are shown in Figure 4.3. They show that the

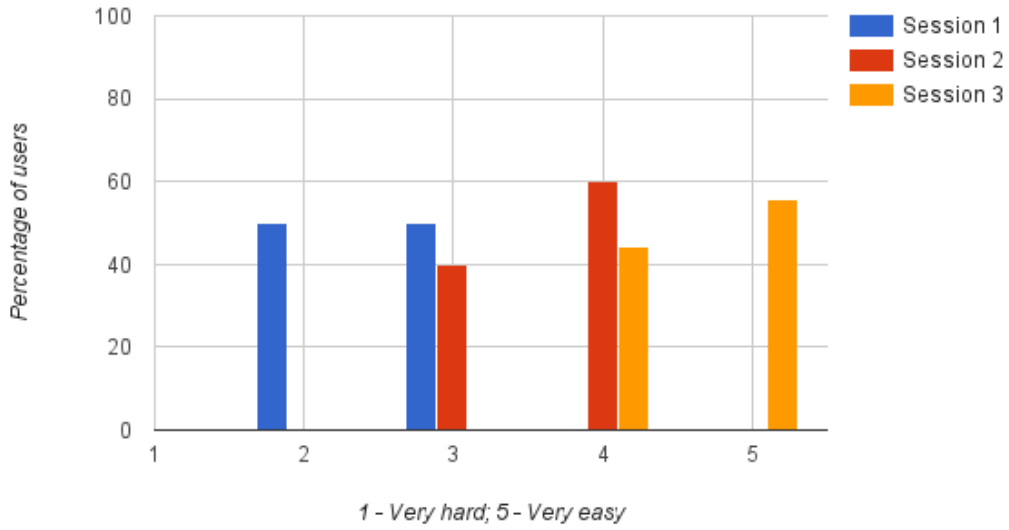


Figure 4.2: Developers' opinion about the difficulty in understanding the system.

developers found the exercises' difficulty to be medium-easy.

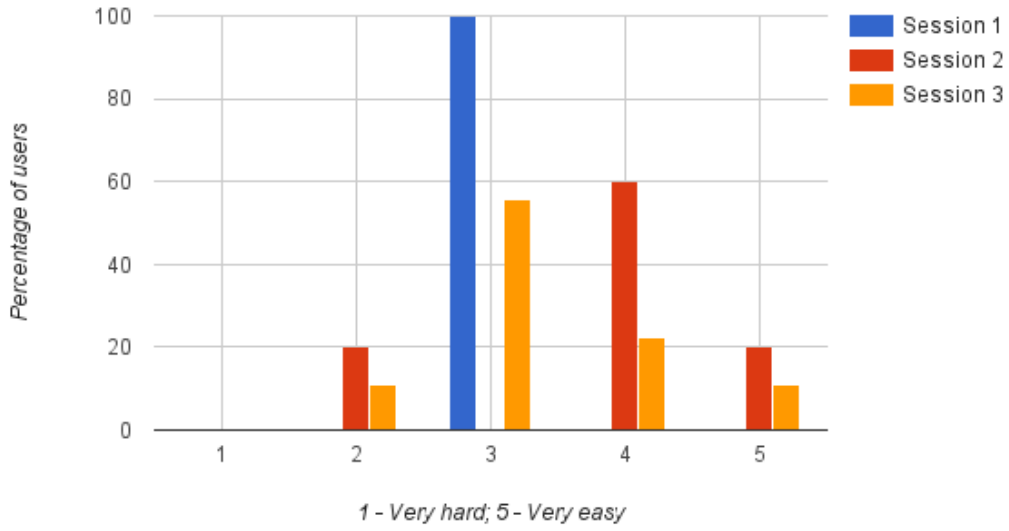


Figure 4.3: Developers' opinion about the difficulty in performing the exercises.

The second question was used to count the number of developers that were able to solve the first exercise. The responses are represented in Figure 4.4. They show that most developers completed the first exercise.

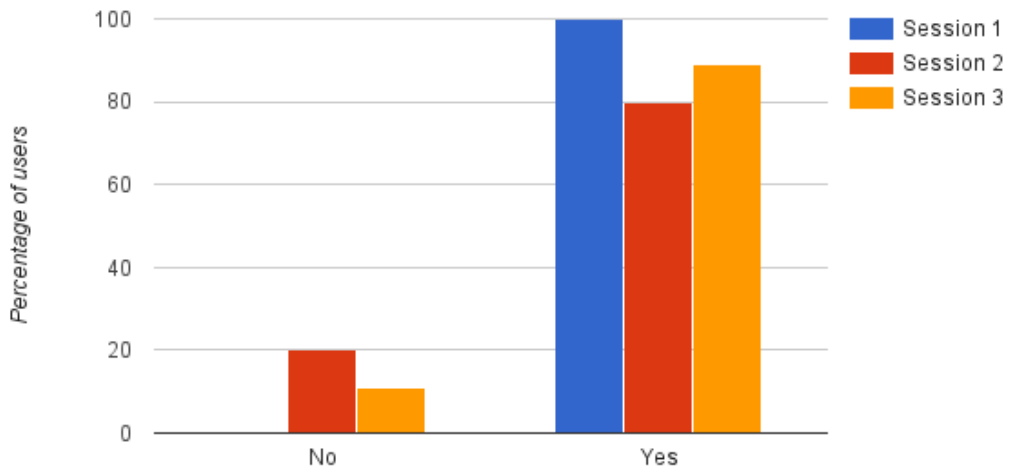


Figure 4.4: Results of the first exercise.

The third question was used to count the number of developers that were able to solve the second exercise. The responses are shown in Figure 4.5. They show that only half the developers were able to solve the second exercise.

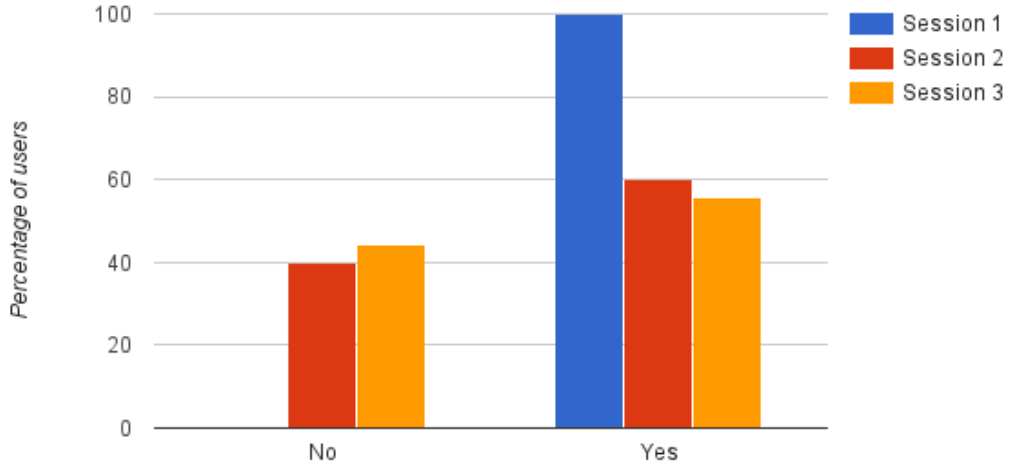


Figure 4.5: Results of the second exercise.

With the results described above, we can conclude that:

- As the number of sessions increased, the percentage of developers that understood the system increased as well.
- The developers found the idea very interesting.
- About half the developers were able to solve both exercises in the time limit, which means they understood the system and the API.

An overall conclusion of these results is that the solution can be extended by developers in practice, thereby improving its usefulness.

4.2 User Interface evaluation (End-user's perspective)

This solution allows end-users to automate tasks that depend on Internet services. We want to understand whether end-users want and are able to create the event-processing rules used to automate tasks.

4.2.1 Methodology

The tests performed with end-users consisted in creating three rules in the Android application. The tests' descriptions are available in Appendix A. The scenario used in the tests is a person looking for job offers, similar to the one described in Chapter 1. The first rule corresponds to sending an email when a new job offer is available. The second rule builds upon the first rule and requires that the job offers' programming language is Java. The third rule builds upon the second rule and requires that the events are grouped in a two week time window. To support these rules, two channels had to be created: the Landing Jobs channel which provides Information Technology (IT) job offers and the Gmail channel, which allows to send emails. Additionally, three EPAs were also implemented: the filter, the aggregate and the higher-than-average. The filter EPA allows to filter events by a certain parameter. The aggregate EPA allows to group events by a certain time period (seconds, minutes, days, weeks, etc). The higher-than-average EPA emits an event when the average value of a list of events is higher than a given value. It was only created to verify that the application supports pattern matching. Each EPA corresponds to each type of EPAs described in Chapter 1. To understand the end-users' opinion about the system, a questionnaire was also created, that is available in Appendix C.

4.2.2 Setup

Each end-user started by reading a short description about the application and, only afterwards, solved the tests. Three metrics were collected during the tests: time to perform the test, number of errors performed and number of help requests. With respect to the first metric, the end-users had 10 seconds to read the description of the test before the time started counting. The expected time for the end-users to complete the first, second and third tests was 120, 150 and 180 seconds respectively. These expectations took into account the fact that these end-users had never used the application before, and correspond to two times the time taken by an experienced user. With respect to the second metric, only serious errors were accounted. For example, when creating a rule, selecting the EC before the EPA is an example of an error that was taken into account. Performing an error when using the Android keyboard to write the name of the rule title is an example of an error that was not taken into account. Regarding the third metric, each time the user requested help, we gave an hint to the correct solution.

After the tests, the end-users had to fill in a questionnaire so we could understand whether they had used similar systems and what they thought about this solution. All tests were performed in the same environment, which was an empty, quiet room in Instituto Superior Técnico (IST). The users were not interrupted during the tests.

4.2.3 Results and Discussion

Due to time constraints, only one session of tests with end-users was performed. The session involved 26 university students, 70% male and 30% female, aged 19 to 26. The results of the session are described below.

As previously referred, one of the metrics collected was the number of errors performed in each exercise. Figure 4.6 shows the average number of errors performed by each end-user in each exercise. As it can be observed, the number of errors performed in the first exercise is lower than the number of errors of the other two exercises (0.4 vs 1.2). This may indicate that when introducing EPAs, the end-users had more difficulty in creating the rules. Another possibility that was observed with a few people is that some end-users expected to first select the EP and the EC, and only afterwards the EPAs. This is not possible in the current version of the Android application.

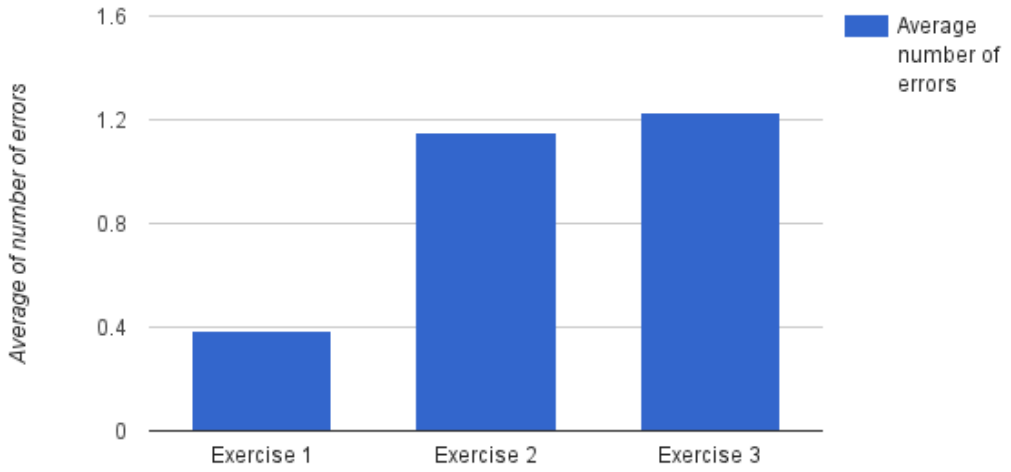


Figure 4.6: The average number of errors performed by the end-users during each exercise.

Another metric collected was the number of help requests received during each exercise. Figure 4.7 shows the average number of help requests received from each end-user during each exercise. In this chart we can see that, when introducing the EPAs, the number of help requests rose, but after the end-users had learned how they work, the number of help requests fell. Still, the number of requests in the first exercise is lower than the other exercises, which may again indicate that end-users were not familiarized with these transformations.

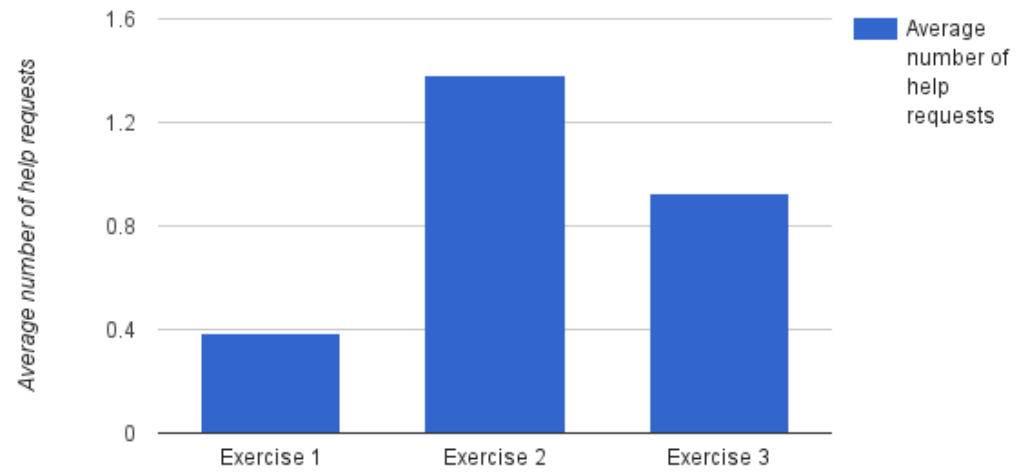


Figure 4.7: The average number of help requests received from the end-users during each exercise.

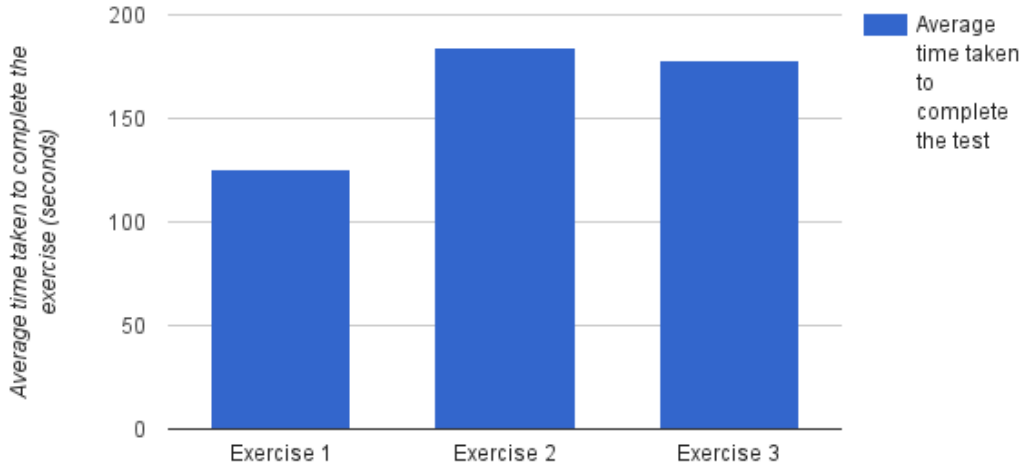


Figure 4.8: The average time taken by the end-users to successfully complete each exercise.

The last metric collected during the tests was the time taken by each end-user to complete each exercise. The results regarding this metric are available in Figure 4.8. They show that, when the end-users first encountered the EPAs, they took some time to think and explore the application. After they had learned how they work, the time fell, as can be observed by comparing the average time taken in the second and third exercises. Also notice that, even though the third exercise required more steps than the second exercise, users took less time to complete it. When comparing these results with the expected results specified above, with exception of the second exercise, which deviates about 30 seconds

from the expected result, the other results are close to the expected values. This fact suggests that the application is easy to use.

During the tests the end-users made some comments that are important to keep in mind because they may lead to overall improvements of the system:

- *"Sorry that I am making so many mistakes with the keyboard."*
- *"Writing the title of the rule is boring."*
- *"The filter should have a list of choices instead of manually typing the name of the parameter of the event."*
- *"I do not like the "When Then Then Then.""*
- *"This application is similar to programming."*
- *"The first screen of the application should have a short tutorial explaining how to create rules."*

One problem that stands out more is the fact that users must type the event parameter in the filter EPA. The implementation of a solution that would suggest the possible choices is feasible. Due to time constraints, this improvement was not implemented. Other comments provide really valuable feedback and some insights about the solution. For example, the comment that compares this solution to programming may be a warning that people do not think this way (When this, then that and then that).

After the exercises, the end-users were asked to fill in a questionnaire. There were two questions that provide more valuable results. The first question was about the usefulness of the idea. The results are shown in Figure 4.9. It shows that all end-users thought the idea was useful.

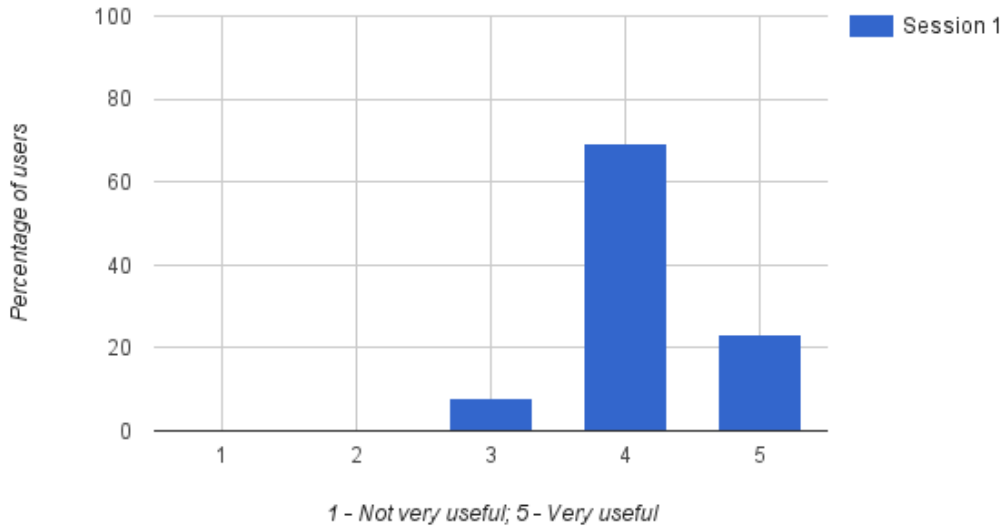


Figure 4.9: End-users' opinion about the usefulness of the idea.

The second question was about the difficulty of the exercises. The results can be observed in Figure 4.10. They show that only 8% of the end-users thought the exercises were hard. This means that most end-users thought the solution was usable.

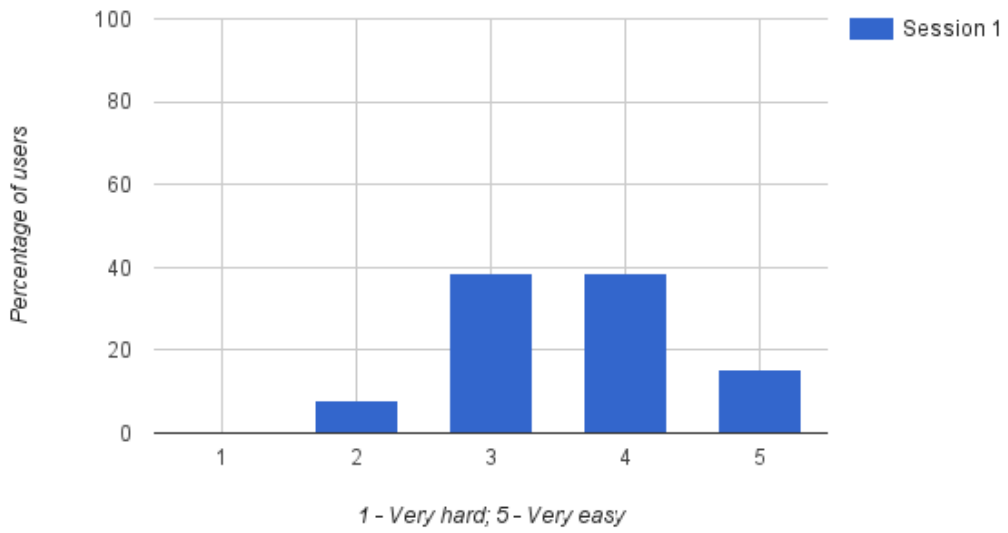


Figure 4.10: End-users' opinion about the difficulty in understanding the system.

With the results described above, we can conclude that:

- End-users must type as least as possible because they can easily make several mistakes just by using the Android keyboard.
- After learning how the application works, end-users needed less time to create the rules and requested help fewer times.
- Most end-users were able to understand the system and use it.
- All end-users liked the idea and only 8% thought it was hard to use.

4.3 Summary

This chapter described the evaluation of the prototype presented in Chapter 3. The prototype was evaluated with developers and end-users, which are the target users of this system. With the results obtained with developers, we can conclude that the system can be easily extended by third-party developers. The quality of the documentation provided with the system is crucial for this task. With the results obtained with end-users, we observe that the mobile application is usable but there is still room for improvement. The results show that EPAs are not obvious to the end-users. However, they also show that, after a short training, end-users have learned how they work and knew how to use them. In conclusion, this assessment was essential to understand what works in the present prototype and what can still be improved in future versions.

Chapter 5

Conclusion

This dissertation addressed the problem of allowing end-users to create event-processing rules to automate tasks involving their personal Internet services. There were two problems with the existing state-of-the-art: the extensibility of the solutions and the expressivity of the rules. The extensibility of the solutions is related to the limited number of Internet services available in the platform. A rule system that allows users to automate tasks should support as many services as possible, otherwise users may not be able to automate the tasks they need because a required service is not supported. The expressivity of the rules is also a problem when users want to create more advanced rules and the system does not allow them. The added complexity of a rule system that allows more expressive rules should still be simple to use. This dissertation proposed a solution that aims to solve the two problems described above.

The solution to the first problem - extensibility - was to allow third-party developers to extend the platform by creating new event providers, event processing agents and event consumers. A prototype that implemented the proposed architecture was created and evaluated with developers. The evaluation concluded that developers found the system easy to extend.

The solution to the second problem - expressiveness - has two parts. The first part is related to the structure of the rules. The existing event processing systems for end-users only allow them to create simples rules that use **one** event provider and **one** event consumer. Instead, our proposed structure consists of **one** event provider, **zero or more** event processing agents and **one** event consumer. This structure is still simple to understand but allows more advanced rules than the previous structure. The second part of the solution is related to the simplicity of the user interface. To achieve this goal, the prototype has several details described in Section 3.4 that simplify the final design. One of the details is that event providers and event consumers are not differentiated in the user interface. The end-users do not really need to know the differences. Another important detail is the process of creating rules, starting with a “WHEN” and then adding “THEN” as they are needed. This way, end-users can better understand the behavior of the rule they are creating. The prototype was also evaluated with end-users and concluded that, despite having some initial difficulties in understanding the event processing agents, end-users were able to understand and create the rules.

Overall, we think this dissertation contributes with an architecture design for a platform that allows end-users to automate tasks that depend on Internet services. A prototype that implements this architecture was created and validated with developers and end-users.

5.1 Future Work

There are several possible paths for enhancing the present solution.

From an end-user perspective, the mobile application could benefit from some improvements. A first improvement could be the creation of a tutorial that teaches users how to create rules in the application. This tutorial would be displayed to the users when they entered the application for the first time. This way, users would learn how to use the application without having to leave it (to read documentation elsewhere). This is a good practice followed by many commercial mobile applications. As a second improvement, the application could allow users to share rules. This would allow users to use rules they have never thought of by themselves, creating a social dimension in the system. Another improvement that could be implemented is the automatic generation of relevant text for the input fields that require the user to type in (e.g. title suggestions for rules). This way, the users would have to type less text when creating a rule. Another interesting improvement would be to support user devices (such as smartphones) because they can provide new event providers and event consumers. For example, it would allow end-users to automate tasks based on their location.

Typically, this type of solution is provided as a Software as a Service (SaaS), i.e. it would run as a service for end-users, hosted on a cloud server. This means that a developer or a company would provide this application for others (end-users) to use. From a service provider perspective, the system could benefit from improved performance that could be achieved by distributing the architecture (database, event processing engine and web server) in several servers. This would allow to support more users and active rules.

Before the system can be used with more sensitive data, a security assessment should also be performed, to understand the type of threats present in the platform. This topic is very important, especially when the system has access to the users' information and to the functionality available in Internet services (e.g. imagine the consequences of an attacker gaining access to the users' email account). The security assessment would allow to understand the type of threats present in the rule engine, as it processes private user information in plain-text and can perform actions on behalf of the user (if it was previously authorized). This could be achieved by creating a STRIDE [24] threat model.

In the near future, end-users should be able to better understand their personal Internet services and to take more advantage of them by putting them working better together. This work is a step forward in that direction, in that it describes an extensible, user-friendly, rule system for connecting Internet services.

Bibliography

[1] W. M. Van Der Aalst, A. H. Ter Hofstede, and M. Weske, "Business process management: A survey," in *Business process management*. Springer, 2003, pp. 1–12.

[2] IFTTT, "IFTTT," accessed on 9 May 2016. [Online]. Available: https://ifttt.com/

[3] D. C. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[4] O. Etzion and P. Niblett, *Event Processing in Action*, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2010.

[5] G. Sharon and O. Etzion, *Event Processing Network - A Conceptual Model*. Technion-Israel Institute of Technology, Faculty of Industrial and Management Engineering, 2007.

[6] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[7] Zapier, "Zapier," accessed on 9 May 2016. [Online]. Available: https://zapier.com/

[8] W. W. Web, "We Wired Web," accessed on 9 May 2016. [Online]. Available: https://wewiredweb.com/

[9] I. E. Technology, "Node-Red," accessed on 9 May 2016. [Online]. Available: http://nodered.org/

[10] D. Hardt, "The OAuth 2.0 authorization framework," 2012.

[11] A. S. Foundation, "Apache Flink," accessed on 9 May 2016. [Online]. Available: http://flink.apache.org/

[12] ——, "Apache Samza," accessed on 9 May 2016. [Online]. Available: http://samza.apache.org/

[13] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets." *HotCloud*, vol. 10, 2010.

[14] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed Stream Computing Platform," *2010 IEEE International Conference on Data Mining Workshops*, pp. 170–177, Dec. 2010.

[15] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy, "Storm@Twitter," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14. ACM, 2014, pp. 147–156.

[16] L. Brenna, A. Demers, J. Gehrke, M. Hong, J. Ossher, B. Panda, M. Riedewald, M. Thatte, and W. White, “Cayuga: A high-performance event processing engine,” in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD ’07. New York, NY, USA: ACM, 2007.

[17] A. Adi and O. Etzion, “The situation manager rule language.” in *RuleML*, vol. 60, 2002, pp. 36–57.

[18] R. Etter, P. D. Costa, and T. Broens, “A rule-based approach towards context-aware user notification services,” in *Pervasive Services, 2006 ACS/IEEE International Conference on*. IEEE, 2006, pp. 281–284.

[19] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik, “Monitoring streams: a new class of data management applications,” in *Proceedings of the 28th international conference on Very Large Data Bases*. VLDB Endowment, 2002, pp. 215–226.

[20] EsperTech, “Esper - Complex Event Processing,” accessed on 9 May 2016. [Online]. Available: http://www.espertech.com/esper/

[21] IBM, “IBM WebSphere,” accessed on 9 May 2016. [Online]. Available: http://www-03.ibm.com/software/products/en/category/operational-decision-management

[22] H. Obweger, J. Schiefer, M. Suntinger, F. Breier, and R. Thullner, “Complex Event Processing Off the Shelf,” in *19th Mediterranean Conference on Control & Automation (MED)*, 2011.

[23] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.

[24] Microsoft, “The stride threat model.” [Online]. Available: https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx

Appendix A

Exercises

In this chapter are available the exercises for developers and the exercises for end-users that were used during their respective evaluations.

Developers' Exercises

The shAPPerd platform is only interesting when there are several event providers and actions. An exercise was created that aims to understand if developers can easily extend the platform.

The exercise is available in the `path/to/shAPPerd-evaluation/exercise` folder and contains a template where you only have to change the TODO sections. Note that the exercise requires that you have previously installed the shAPPerd platform.

To verify that your solution to the exercise is correct, execute the following commands in the command-line:

```
cd /path/to/shAPPerd-evaluation/exercise
mvn clean test
```

You should see the following output:

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building exercise 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]

...

-------------------------------------------------------
 T E S T S
 -------------------------------------------------------

...

Results :

Tests in error:
  testExecute(exercise.TestLogFile)
    testGetVariables(exercise.TestLogFileDescription)
      testListen(exercise.TestRSSFeed)
        testGetVariables(exercise.TestRSSFeedDescription)
          testGetEventParameters(exercise.TestRSSFeedDescription)

Tests run: 5, Failures: 0, Errors: 5, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 4.152 s
[INFO] Finished at: 2015-09-16T21:08:41+01:00
[INFO] Final Memory: 16M/143M
[INFO] ------------------------------------------------------------------------
[ERROR] Failed to execute goal
```

```
org.apache.maven.plugins:maven-surefire-plugin:2.12.4:test (default-test) on
project exercise: There are test failures.
[ERROR]
[ERROR] Please refer to
/Users/fury/Github/shAPPerd-evaluation/exercise/target/surefire-reports for the
individual test results.
[ERROR] -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read
the following articles:
[ERROR] [Help 1]
http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
```

As you implement more methods, you should pass more tests.

Task 1

The first task consists of creating an RSSFeed event provider. You can learn about RSS here. To implement this event provider, you have to implement the classes `RSSFeed` and `RSSFeedDescription`. Each event emitted by the event provider represents an entry/item of the RSS feed.

This event provider will receive one variable named `url`, which is the URL of the RSS feed to be used. The hint of the url is `http://www.lighttable.com/atom.xml`. Each event contains the parameters `title` and `link`. The listen method must read the RSS feed of the given url and emit an event for each item available in the RSS feed. To avoid implementing an RSS feed reader, the package de.vogella.rss available in the template already implements an RSS feed reader that reads RSS feeds from an URL.

Task 2

The second task consists of creating an action that logs messages to a file. For this, you have to implement the classes `LogFile` and `LogFileDescription`. This action will append (insert in the end) a given message to a given file.

The single variable is `filename`. The hint of the filename is `lighttable_feed.txt`. The execute method must append the event and the character `\n` to the file with the given filename. To convert the event to a string, use event.toString().

End-users' Exercises

The exercises consist on creating rules in the shAPPerd Android application.

1st Rule

Send an email to `fury252@gmail.com` with the subject `New job offer` when a `new job offer` is available in `Jobbox`.
`Every parameter` of the job offer is `interesting`.
The rule should be called `Receive an email with the latest job offers.`

2nd Rule

Send an email to `fury252@gmail.com` with the subject `New JAVA job offer` when a `new job offer` is available in `Jobbox`, with the added condition:

- the `programming_languages` parameter must contain `Java`

The `single interesting parameter` of the job offer is `city`.
The rule should be called `Receive an email with the latest JAVA job offers.`

3rd Rule

Send an email to `fury252@gmail.com` with the subject `New JAVA job offers in the last two weeks` when a `new job offer` is available in Jobbox, with the added conditions:

- the `programming_languages` parameter must contain `Java`
- the `job offers` must be grouped by 2 weeks

`Every parameter` of the job offer is `interesting`.
The rule should be called `Receive an email every two weeks with the latest JAVA job offers.`

Appendix B

Developers' Documentation

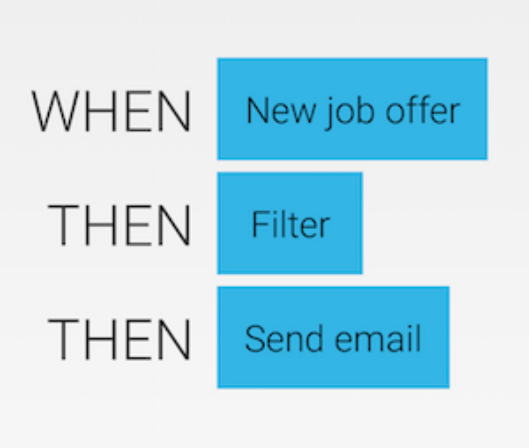
In this chapter is available the documentation that explains developers how to install, use and extend the solution.

shAPPerd

shAPPerd is an application that allows users to connect services available in the Internet, such as Landing.Jobs and Gmail, using rules with the form '**WHEN** event **THEN** action'.

Example

Imagine that you want to receive an email for every new job offer that appears on the Landing.Jobs site (a tech jobs marketplace), with the added condition that the job offer must be related to Java. With shAPPerd, you can create a rule such that, when a new Java job offer appears, an email is sent to your email address.



Documentation

This section explains the fundamental building blocks in order to understand how shAPPerd works.

Concepts

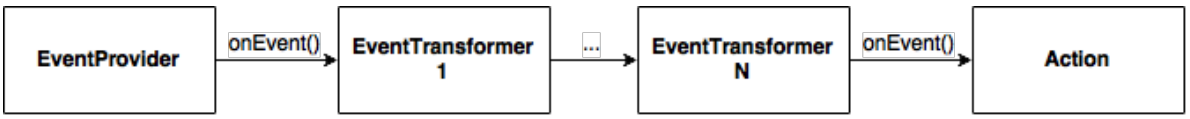
Channel

A **channel** is an entity that represents a real service. Examples of services including Landing.Jobs, Gmail, Dropbox, etc. The channel is responsible for grouping together related event providers and actions.

Currently there are only two channels available in shAPPerd (Landing.Jobs and Gmail), but developers can create new channels and integrate them in the platform.

Rule

Rules are composed by three elements: **event providers**, **event transformers** and **actions**. When a rule is installed, the events start in the event provider, then are transformed by the event transformers, and finally trigger the action.



Event provider

An **event provider** is responsible for emitting events. There are two types of event providers:

- Event Listeners, which are always watching for new events (polling);
- Event Handlers, which are executed when an HTTP POST with the rule id is performed.

An event provider specifies the structure of the events it emits (the event parameters). It may also declare the variables that are required before execution. A possible example is the URL of an RSS feed. An RSS feed reader must know the URL before it gets the information of the feed.

Event transformer

An **event transformer** is responsible for transforming events. One example is the filter event transformer that filters the events it receives. An event transformer can also have variables like the event provider. Consider the filter event transformer. It can have as parameters the name and the value of the parameter it wants to filter in the incoming events.

As you may have noticed, there are certain cases where an event emitted by an event provider does not trigger the action. Consider the case where an event is filtered by an event transformer because the value of a given parameter is not the expected one.

Action

An **action** is responsible for executing a specific procedure. An action may also declare variables whose values are required for its usage. An example of an action is **send email** of the Gmail channel.

Event

An **event** is just a mapping of names to values.

Installation

This section describes how to install shAPPerd in your computer and how to use it.

Requirements

Before you install shAPPerd, you must have available in the PATH environment variable the following programs:

- Java Development Kit (JDK) 1.8+
- Maven 3+

Installation

You can download the shAPPerd platform here.

To install the shAPPerd platform in your system, unzip the file and execute the following commands in the command-line:

```
cd /path/to/shapperd-evaluation/
mvn clean source:jar install
```

You should see the following output:

```
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Build Order:
[INFO]
[INFO] shapperd-api
[INFO] jobbox-channel
[INFO] terminal-printer-channel
[INFO] gmail-channel
[INFO] shapperd
[INFO] shapperd-parent
[INFO]

...

[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] shapperd-api ....................................... SUCCESS [  3.095 s]
[INFO] jobbox-channel ..................................... SUCCESS [  0.280 s]
[INFO] terminal-printer-channel ........................... SUCCESS [  0.123 s]
[INFO] gmail-channel ...................................... SUCCESS [  0.464 s]
[INFO] shapperd ........................................... SUCCESS [  1.510 s]
[INFO] shapperd-parent .................................... SUCCESS [  0.012 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 6.038 s
[INFO] Finished at: 2015-09-16T15:57:20+01:00
[INFO] Final Memory: 28M/227M
[INFO] ------------------------------------------------------------------------
```

Usage

The standard way to use shAPPerd is to start the server in a machine, and then use the Android application to create rules and view the existing channels in the platform.

To start the server, execute the following commands in the command-line (assuming you have already installed the shAPPerd platform):

```
cd /path/to/shapperd-evaluation
mvn exec:java -pl shapperd
```

You should see the following output:

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building shapperd 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- exec-maven-plugin:1.4.0:java (default-cli) @ shapperd ---
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in
[jar:file:/Users/fury/.m2/repository/org/slf4j/slf4j-simple/1.7.7/slf4j-simple-1.7.
7.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in
[jar:file:/Users/fury/.m2/repository/ch/qos/logback/logback-classic/1.0.13/logback-
classic-1.0.13.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.SimpleLoggerFactory]
[Thread-1] INFO spark.webserver.SparkServer - == Spark has ignited ...
[Thread-1] INFO spark.webserver.SparkServer - >> Listening on 0.0.0.0:8888
[Thread-1] INFO org.eclipse.jetty.server.Server - jetty-9.0.2.v20130417
[Thread-1] INFO org.eclipse.jetty.server.ServerConnector - Started
ServerConnector@2ad0490f{HTTP/1.1}{0.0.0.0:8888}
```

Now that the server is running, you just need to install the Android application in your smartphone and run it.

Before installing the Android app, you must specify where the server is located. To do this, execute the following commands:

```
cd
/path/to/shapperd-evaluation/shapperd-android/ShAPPerd/ShapperdAPIClient/src/main/a
ssets;
cp server-conf-template.json server-conf.json;

Now modify the server-conf.json file and replace the value "0.0.0.0"
with your internal IP address if the computer running the server and the
smartphone are in the same network.
```

To install the Android application in your smartphone, we recommend using Android Studio. The shapperd-android folder contains all the source code of the Android application.

Extend shAPPerd

This document explains how you can create new channels and integrate them with the platform. The Jobbox channel will be used as an example during the explanation.

The source code used below is available in the folder `/path/to/shAPPerd-evaluation/shapperd-api/src/main/java/pt/tecnico/shapperd`. The javadoc is available here.

The source of the Jobbox channel is available in the folder `/path/to/shAPPerd-evaluation/channels/jobbox-channel/src/main/java/io/jobbox/channel`.

Channel

A **channel** represents a real service. In shAPPerd, this means that it groups related event providers (event listeners and event handlers) and actions.

A channel can have a service dependency. This means that, in order to use the channel, the user must first resolve the service dependency. Consider the Gmail channel. One of the possible actions it may provide is the send email action. For this action to work, the user must first have a Gmail account. This is the purpose of the service dependency. It specifies that the user must first allow the application to access its Gmail account before using any of its functionality.

```
/**
 * A channel groups similar event providers and actions.
 */
public interface Channel {
    /**
     * @return the name of the channel.
     */
    public String getName();

    /**
     * @return the description of the channel.
     */
    public String getDescription();

    /**
     * @return the event listeners available in this channel.
     */
    public EventListenerDescription[] getEventListeners();

    /**
     * @return the event handlers available in this channel.
     */
    public EventHandlerDescription[] getEventHandlers();

    /**
     * @return the actions available in this channel.
     */
```

```
    public ActionDescription[] getActions();

    /**
     * @return the service dependencies that must be fulfilled before using any
EventListener|EventHandler|Action
     * available in this channel.
     */
    public ServiceDependency[] getServiceDependencies();
}
```

The existing implementation of the Jobbox channel only provides one event listener, the NewJobOffer.

```
public class JobboxChannel implements Channel {
    @Override
    public String getName() {
        return "Jobbox";
    }

    @Override
    public String getDescription() {
        return "Tech jobs marketplace";
    }

    @Override
    public EventListenerDescription[] getEventListeners() {
        return new EventListenerDescription[] {
                new NewJobOfferDescription()
        };
    }

    @Override
    public EventHandlerDescription[] getEventHandlers() {
        return new EventHandlerDescription[0];
    }

    @Override
    public ActionDescription[] getActions() {
        return new ActionDescription[0];
    }

    @Override
    public ServiceDependency[] getServiceDependencies() {
        return new ServiceDependency[0];
    }
}
```

As you may have noticed, a channel does not return the event providers but their descriptions. The same applies to actions. This is because a description describes some properties of the event provider, such as the name, the description, the variables, hints for possible values for the variables and the event parameters. This information is used by the platform to show the user how the event provider works.

```
public interface EventProviderDescription extends Serializable {
    public String getName();
    public String getDescription();

    /**
     * @return the variables used by the event listener.
     */
    public String[] getVariables();

    /**
     * @return a hint of a possible value for the variable.
     */
    public Map<String, String> getVariableHint();

    /**
     * @return the parameters that form the events returned by the event listener.
     */
    public String[] getEventParameters();
}

public interface EventListenerDescription extends EventProviderDescription {
    /**
     * @return an event listener that implements this description.
     */
    public EventListener create(Map<String, Service> services, EventHandler
callback, Map<String, String> variables);
}
```

As you can see, the NewJobOfferDescription provides a user-friendly name and description. In this case, it does not require any variables, so its implementation returns an empty array.

```
public class NewJobOfferDescription implements EventListenerDescription {
    @Override
    public String getName() {
        return "New job offer";
    }

    @Override
    public String getDescription() {
        return "Keep up with the best IT job offers.";
    }

    @Override
    public String[] getVariables() {
        return new String[0];
    }

    @Override
    public Map<String, String> getVariableHint() {
        return new HashMap<>();
    }

    @Override
    public String[] getEventParameters() {
```

```
        return new String[] {
                "city",
                "reward",
                "programming_languages",
                "type"
        };
    }

    @Override
    public EventListener create(Map<String, Service> services, EventHandler
callback, Map<String, String> variables) {
        return new NewJobOffer(callback, services, variables);
    }
}
```

The create method is responsible for returning the right implementation of the event provider (either EventListener or EventHandler). The services argument contains the service dependencies resolved. In this case, the map will be empty because the channel declared zero dependencies. The callback argument is used to emit events. The variables argument corresponds to the values specified by the user. In this case, it will also be an empty map because the description declared zero variables.

An event listener must implement a method called listen that is called by the system when the rule is installed. When its implementation contains new events to emit, it calls the method onEvent of the callback passed in the constructor.

```
/**
 * An EventListener is the mechanism used to receive events from a source
synchronously.
 */
public abstract class EventListener extends EventProvider implements Serializable {
    public EventListener(EventHandler callback, Map<String, Service> services,
Map<String, String> variables) {
        super(callback, services, variables);
    }

    /**
     * Starts listening for events and when an event occurs, it calls the
EventReceiver#onEvent(event).
     */
    public abstract void listen();
}
```

As the name of the description suggested, the implementation of the event listener is called NewJobOffer.
The current implementation reads a json file containing job offers and, for each job offer in the file, it emits one event.

Internally, shaPPerd uses a data format protocol consisting of java.util.Map and java.util.List to structure information. This means that if the parameters of the event contain a list or a map of

values, the recommended strategy is that the objects that hold them should implement either the java.util.List interface of the java.util.Map interface.
This data format protocol is required to print the events in a user-friendlier way.

```
/**
 * Due to having problems related to the SSL certificate of the Jobbox service, the
events emitted are the ones stored
 * in the file resources/offers.json.
 */
public class NewJobOffer extends EventListener {
    public NewJobOffer(EventHandler callback, Map<String, Service> services,
Map<String, String> variables) {
        super(callback, services, variables);
    }

    private static Collection<Object> toCollection(JSONArray json) {
        List<Object> content = new ArrayList<>();
        for (int i = 0; i < json.length(); i++) {
            Object val = json.get(i);
            if (val instanceof JSONArray) {
                content.add(toCollection((JSONArray) val));
            } else if (val instanceof JSONObject) {
                content.add(toMap((JSONObject) val));
            } else {
                content.add(val);
            }
        }
        return content;
    }

    private static Map<String, Object> toMap(JSONObject json) {
        Map<String, Object> content = new HashMap<>();
        for (String key : json.keySet()) {
            Object val = json.get(key);
            if (val instanceof JSONArray) {
                content.put(key, toCollection((JSONArray) val));
            } else if (val instanceof JSONObject) {
                content.put(key, toMap((JSONObject) val));
            } else {
                content.put(key, val);
            }
        }
        return content;
    }

    public void listen() {
        final String filename = "/offers.json";

        try {
            final InputStream is = this.getClass().getResourceAsStream(filename);
            JSONArray offers = JSONReader.readJsonArray(is);

            for (int i = 0; i < offers.length(); i++) {
                JSONObject jsonEntry = offers.getJSONObject(i);
                Event event = new Event(toMap(jsonEntry));
```

```
                    getCallback().onEvent(event);
                }
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
    }
```

Appendix C

Questionnaires

In this chapter are available the questionnaires for developers and the questionnaires for end-users that were used during their respective evaluations.

Sistema de regras shAPPerd - Avaliação com programadores

E se as suas Apps preferidas pudessem colaborar umas com as outras?
O shAPPerd pode ajudar!

Obrigado pela disponibilidade para ajudar neste projeto de investigação.
Os resultados contribuirão para facilitar a vida das pessoas com novas facilidades de utilização das suas apps.

Este inquérito destina-se a perceber se o sistema shAPPerd pode ser facilmente estendido por programadores, para oferecer novas capacidades ao sistema.

O tempo previsto para a atividade são 45 minutos e será necessário aplicar os seus conhecimentos de programação (nada de muito difícil, esperamos) :)

A documentação estará em inglês... no problem? Great :)

Pedimos que faça tudo de seguida, com o mínimo de interrupções possível.

Agradeço desde já a pela disponibilidade :)

* Required

Informação Pessoal

1. **Idade** *

.....

2. **Sexo** *
   Mark only one oval.

   - ◯ Masculino
   - ◯ Feminino

Informação técnica

3. **Já programas há quanto tempo?** *
   Mark only one oval.

   - ◯ Nunca programei.
   - ◯ Há menos de 1 ano.
   - ◯ Entre 1 e 2 anos.
   - ◯ Entre 2 e 3 anos.
   - ◯ Entre 3 e 5 anos.
   - ◯ Há mais de 5 anos.

4. **Há quantos anos programas em Java?** *
Mark only one oval.

- ◯ Nunca programei em Java.
- ◯ Há menos de 1 ano.
- ◯ Entre 1 e 2 anos.
- ◯ Entre 2 e 3 anos.
- ◯ Há mais de 3 anos.

5. **Alguma vez utilizaste a ferramenta Maven?** *
Mark only one oval.

- ◯ Sim
- ◯ Não

Automatização de tarefas

6. **Já utilizaste alguma aplicação de automatização de tarefas (tipo IFTTT)?** *
Mark only one oval.

- ◯ Sim
- ◯ Não

7. **Se sim, qual?**

.....

8. **Já fizeste um script para automatizar um dado procedimento?** *
Mark only one oval.

- ◯ Sim
- ◯ Não

9. **Se sim, que procedimento é que automatizaste?**

.....

.....

.....

.....

.....

Utilização de serviços e dispositivos

10. **Quais destes serviços utilizas regularmente?** *
    Check all that apply.

- [ ] Facebook
- [ ] Twitter
- [ ] Instagram
- [ ] LinkedIn
- [ ] Pocket
- [ ] Outlook Calendar
- [ ] Outlook Mail
- [ ] Gmail
- [ ] RSS
- [ ] Dropbox
- [ ] Landing Jobs (ex-Jobbox)
- [ ] Evernote
- [ ] Fitbit
- [ ] Nike+
- [ ] Other: .....

11. **Quais destes dispositivos utilizas diariamente?** *
    Check all that apply.

- [ ] Computador
- [ ] Smartphone
- [ ] Tablet
- [ ] Smartwatch
- [ ] Other: .....

Avaliação do sistema shAPPerd

Nesta secção pretende-se avaliar se o sistema shAPPerd é de fácil compreensão e utilização, e ainda se é facilmente estendido.

Introdução

Uma descrição do sistema shAPPerd e de como o instalar encontra-se disponível neste link: https://github.com/furypt/shAPPerd/blob/master/README.md

Preencha as seguintes perguntas apenas depois de leitura dessa introdução.

12. **O que achaste do sistema?** *
    Mark only one oval.

	1	2	3	4	5	
pouco útil	◯	◯	◯	◯	◯	muito útil

13. **Achaste o sistema complicado de entender?** *
Mark only one oval.

	1	2	3	4	5	
difícil compreensão	◯	◯	◯	◯	◯	fácil compreensão

14. **Comentários sobre o sistema**

.....

.....

.....

.....

.....

Extensão do sistema

O sistema shAPPerd apenas se torna interessante quando tem muitos eventos e acções disponíveis. Assim sendo, deve ser facilmente estendido. Para melhor perceberes como podes estender o sistema, lê este documento (https://github.com/furypt/shAPPerd/blob/master/EXTEND-SHAPPERD.md).

De forma a avaliar se o sistema é facilmente extensível, está disponível um exercício neste link (https://github.com/furypt/shAPPerd/blob/master/DEVELOPER-EXERCISES.md), onde terás de criar um novo evento e uma nova acção. O exercício foi adaptado para ser feito em 30 minutos.

Assim sendo, peço que reservem os próximos 30 minutos para fazer o exercício (um novo evento e uma nova acção).

O exercício deve terminar ao fim de 30 minutos, mesmo que não tenha sido terminado. Depois é só responder ao questionário e obrigado pela ajuda!

15. **Que tarefas conseguiste fazer correctamente dentro do tempo esperado?**
Check all that apply.

- ☐ Tarefa 1
- ☐ Tarefa 2

16. **Entre 1 e 5, quão difícil achas a utilização o sistema?** *
Mark only one oval.

	1	2	3	4	5	
fácil	◯	◯	◯	◯	◯	difícil

17. **Comentários sobre o sistema**

.....

.....

.....

.....

.....

Conclusão

18. **Estarias interessado em preencher novos inquéritos?** *

Mark only one oval.

- ◯ Sim
- ◯ Não

Sistema de regras shAPPerd - Avaliação com utilizadores

* Required

Informação pessoal

1. **Idade** *

2. **Sexo** *
   Mark only one oval.
   - Masculino
   - Feminino

Automatização de tarefas

3. **Já utilizaste alguma aplicação de automatização de tarefas (tipo IFTTT)?** *
   Mark only one oval.
   - Sim
   - Não

4. **Se sim, qual?**

Utilização de serviços e dispositivos

5. **Qual destes serviços utilizas regularmente?**
   Check all that apply.

   - [ ] Facebook
   - [ ] Twitter
   - [ ] Instagram
   - [ ] LinkedIn
   - [ ] Pocket
   - [ ] Outlook Calendar
   - [ ] Outlook Mail
   - [ ] Gmail
   - [ ] RSS
   - [ ] Dropbox
   - [ ] Landing Jobs (ex-Jobbox)
   - [ ] Evernote
   - [ ] Fitbit
   - [ ] Nike+
   - [ ] Other: .....

6. **Quais destes dispositivos utilizas diariamente?**
   Check all that apply.

   - [ ] Computador
   - [ ] Smartphone
   - [ ] Tablet
   - [ ] Smartwatch
   - [ ] Other: .....

Avaliação da aplicação Android

7. **O que achaste do sistema?** *
   Mark only one oval.

	1	2	3	4	5	
pouco útil	◯	◯	◯	◯	◯	muito útil

8. **Achaste o sistema complicado de entender?** *
   Mark only one oval.

	1	2	3	4	5	
difícil compreensão	◯	◯	◯	◯	◯	fácil compreensão