Extensible User-friendly Rule System for connecting Internet Services

André Filipe Pereira Rodrigues
Instituto Superior Técnico
Universidade de Lisboa
Av. Rovisco Pais, 1049-001 Lisboa, Portugal
Email: andre.pereira.rodrigues@tecnico.ulisboa.pt

***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 work 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.**

***Index Terms*—End-User Programming, Trigger-Action Programming, Event Processing, Task Automation, Internet Services**

I. 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.

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. It provides an easy-to-use interface designed for end-users, where they 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.

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]. 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

[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.

services provide both data and functionality that can be used in task automation, such as as EPs and ECs. 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[3] could be used as an email provider.

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.

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], [6], [7], 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 [8] that support more EPAs are designed for developers. 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. 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 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.

A. Contribution

The contribution of this work 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.

[3]Gmail - https://mail.google.com. Last accessed on 9 May 2016.

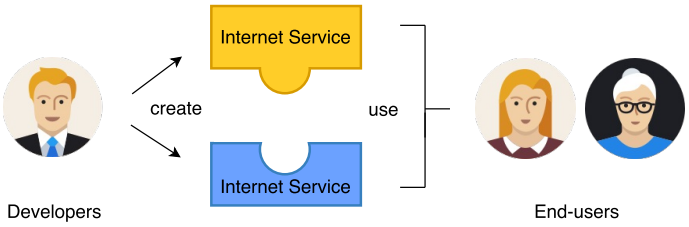

Fig. 1. The roles of developers and end-users in the rule platform.

II. Background

This section starts by presenting a model for Internet services and explains why they are useful for automating tasks. Then explains how EPEs work and what are the differences between them. Finally, it explains how existing systems allow end-users to automate tasks using Internet services.

A. 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.

A survey was performed to understand the type of information and functionality available in Internet services, and how can other applications access user information available in 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) and functionality (e.g. send email). The second is that most Internet services use OAuth [9], an open and secure authorization protocol that allows users to authorize applications to access their information available in an Internet service. 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.

B. Event Processing Engines

EPEs allow users to specify EPNs, which are composed by EPs, EPAs and ECs. The events start at the EPs, which

are then sent to the EPAs that transform them, and finally are consumed by the ECs. When comparing EPEs, one of the most important aspects is 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) [10]. The existing systems used by end-users have the form "IF event THEN action".

C. Rule systems that connect Internet services

This work focuses on systems that allow end-users to automate tasks that depend on Internet services. IFTTT is one of those systems, in that it allows end-users to connect events to actions, as in "If event Then action" [2]. 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[4]). 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.

III. Solution

We are now describing a solution we called shAPPerd[5], 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 Section I. These requirements were validated with the evaluation described in Section IV.

A. Architecture Overview

The solution is divided in three parts: Internet Services, Backend and Mobile Frontend. It is represented in Figure 2. 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).

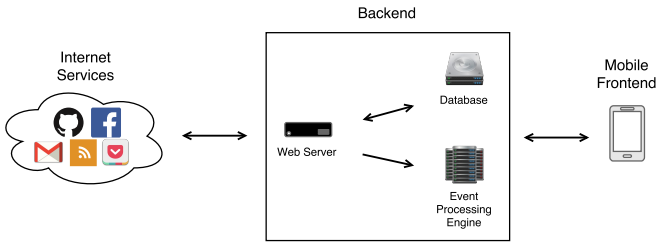

Fig. 2. The three parts that compose the solution: Internet Services, Backend and Mobile Frontend.

A prototype was created that implemented this architecture. It is open-source and available in Github[6], which is an online repository hosting service. The programming language used in the implementation is Java 8[7]. To build and execute the prototype, Maven 3[8] was used. The details of the implementation are explained in the rest of this section.

B. Internet Services

To allow the creation of rules that depend on user information available in Internet services, the prototype supports the OAuth protocol. It 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.

C. Backend

The backend component is responsible for storing the information, communicating with both the Internet services and the mobile frontend and to make the rules work as expected.

One of the decisions made during the development of the backend was concerned about the format of the rules. 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

[4]Dropbox - https://www.dropbox.com. Last accessed on 9 May 2016.

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

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

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

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

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. 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.

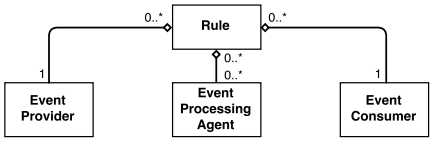

Fig. 3. The rule representation used by the solution.

Another important decision we had to make was about how can end-users use the information available in the events. One approach is to allow the user to refer to the parameters of the event. This approach requires a special syntax to know when the user is referring to a parameter of the event and to know about the parameters of the events. 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). 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)[9], 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.

D. Mobile Frontend

The mobile frontend is the user interface that end-users use to automate tasks. Below is explained what should be available in a user interface for this type of applications, and how the prototype implements this functionality.

The features that should be available in the user interface are:

- Login - functionality like creating rules or view installed rules varies for each user, and, therefore, there should be a mechanism to distinguish users.
- View Channels - this feature promotes explorability because it allows the user to see which Internet services are supported and what they allow him to do.
- Create Rule - this is the central feature of the application. It involves selecting the event provider, the event processing agents and the event consumer.
- View Rules - the user should not have to memorize which rules he has previously created.
- Enable Rule - the user should have the means to enable or disable a rule whenever he wants.
- Delete Rule - the user should also have the means to delete a previously created rule.

A prototype of an Android[10] application was created that implemented these features. A screenshot of the application can be viewed in Figure 4. The Android application is divided in two parts: the Representational State Transfer (REST) API client that communicates with the shAPPerd backend and the user interface.

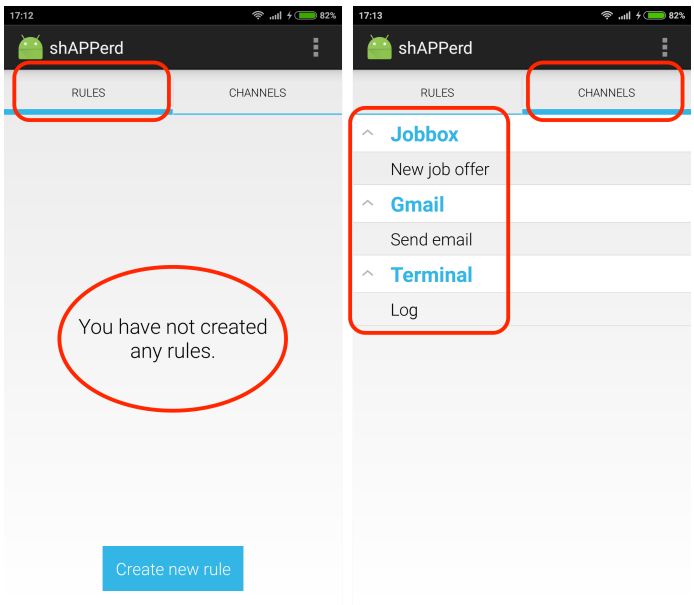

Fig. 4. The Android application.

The user interface contains key details that make the application easier to use. One of them is that there is no distinction between event providers and event consumers. This detail can be observed in the right screen of Figure 4. This is important because most users do not know what are event providers and event consumers and, in fact, they do not really need to know what they are. They just want to know what the system can do. Another key detail, that can be observed in Figure 5 is that, due to the usage of the **WHEN** and the **THEN** keywords, the rules almost read like plain english.

IV. Evaluation

This section explains the approach followed to verify whether our prototype solves the problems described in Sec-

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

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

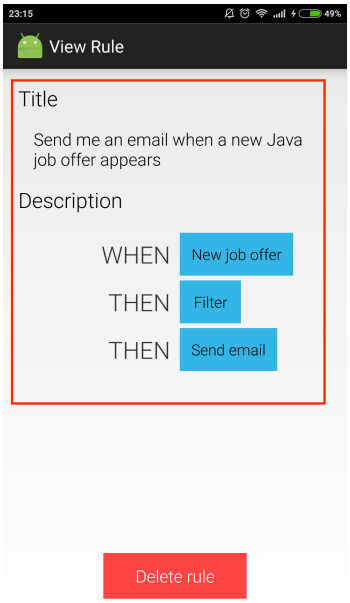

Fig. 5. A rule in the Android application.

tion I, 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 parts: developers and end-users. Each part 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.

A. 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.

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 EP to be implemented is an Rich Site Summary (RSS) feed reader[11] that, given an Uniform Resource Locator (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 Github[12]. 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 was used to learn about the developers' thoughts on the solution and whether they had already used similar systems.

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.

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 6. From the figure, it can be concluded that most developers thought the idea was useful.

Another question was asked to determine whether the developers had understood the system. The responses can be observed in Figure 7. The results show that, as the documentation improved, the developers' understanding of the system also improved.

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

[12]shAPPerd Documentation in Github - https://github.com/furypt/shAPPerd-evaluation/blob/master/README.md. Last accessed on 9 May 2016.

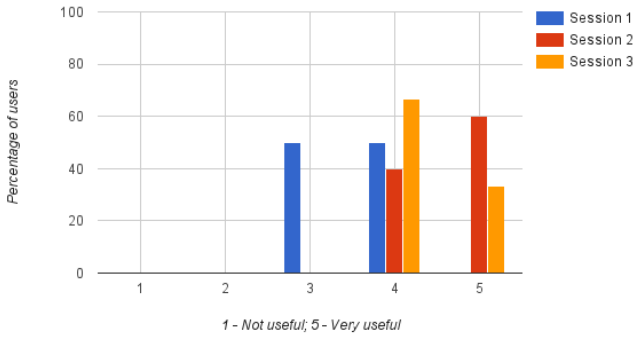

Fig. 6. Developers' opinion about the usefulness of the idea.

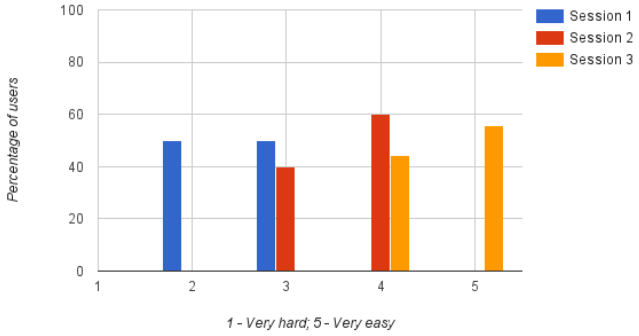

Fig. 7. Developers' opinion about the difficulty in understanding the system.

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 8. They show that the developers found the exercises' difficulty to be medium-easy.

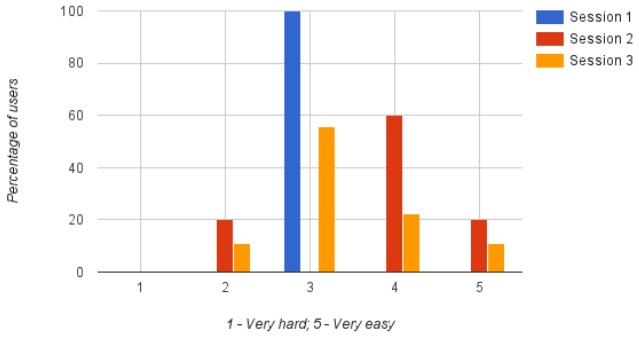

Fig. 8. Developers' opinion about the difficulty in performing the exercises.

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 9. They show that most developers completed the first exercise.

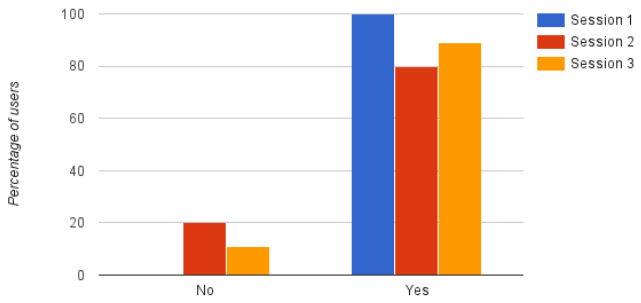

Fig. 9. Results of the first exercise.

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 10. They show that only half the developers were able to solve the second exercise.

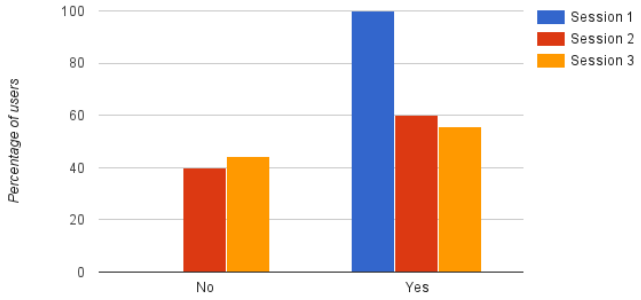

Fig. 10. Results of the second exercise.

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.

B. 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.

1) Methodology: The tests performed with end-users consisted in creating three rules in the Android application. The scenario used in the tests is a person looking for job offers, similar to the one described in Section I. 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 Section I. To understand the end-users' opinion about the system, a questionnaire was also created.

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.

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 11 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.

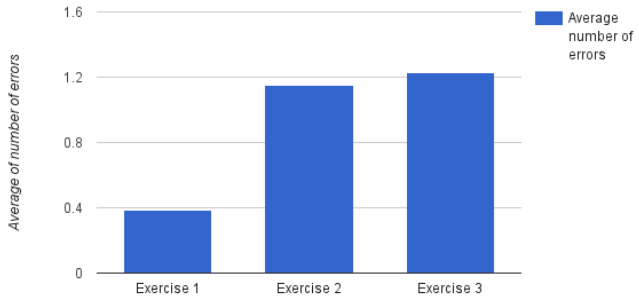

Fig. 11. The average number of errors performed by the end-users during each exercise.

Another metric collected was the number of help requests received during each exercise. Figure 12 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.

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 13. 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.

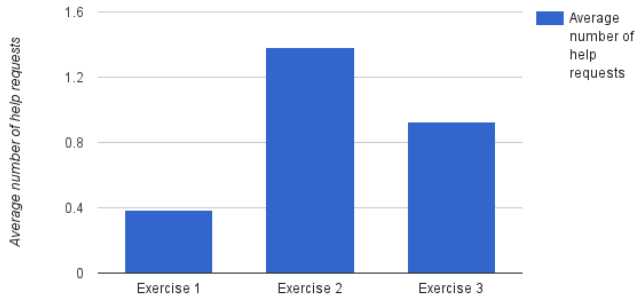

Fig. 12. The average number of help requests received from the end-users during each exercise.

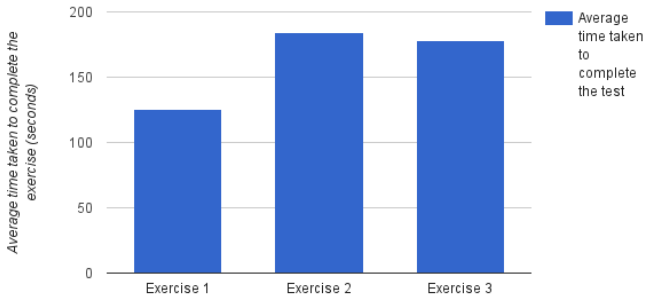

Fig. 13. The average time taken by the end-users to successfully complete each exercise.

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 14. It shows that all end-users thought the idea was useful.

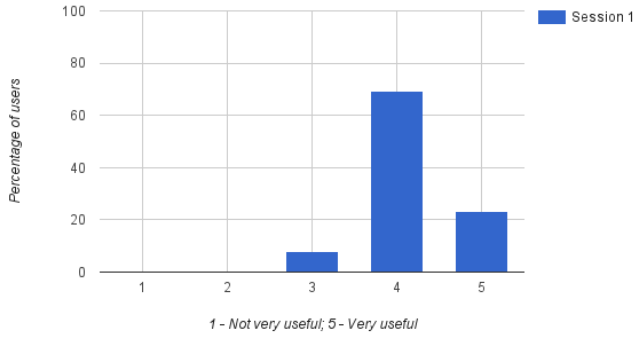

Fig. 14. End-users' opinion about the usefulness of the idea.

The second question was about the difficulty of the exercises. The results can be observed in Figure 15. 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.

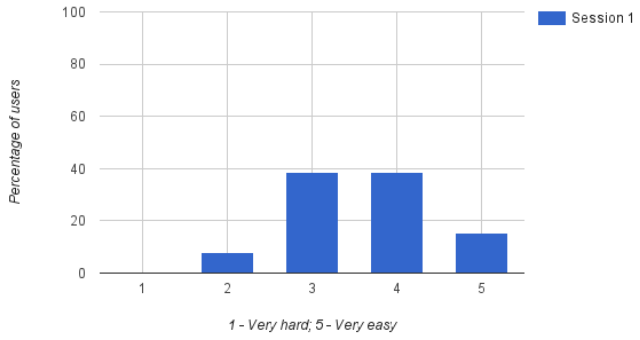

Fig. 15. End-users' opinion about the difficulty in understanding the system.

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.

V. Conclusion

This work 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 solution to this problem 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 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. The solution to this problem 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 key details 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 they were able to understand and create the rules.

A. 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. As a second improvement, the application could allow users to share rules. 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). 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.

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 [11] threat model.

In the 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.

References

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

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

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

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

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

[11] Microsoft, "The STRIDE Threat Model." [Online]. Available: https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx