TRABAJO FIN DE GRADO

Título:	Diseño e implementación de un Sistema Semántico de Automatización de Tareas para Proyectos Ágiles de Desarrollo Software
Título (inglés):	Design and Implementation of a Semantic Task Automation System for Agile Software Project Environments
Autor:	Javier Díez Martínez
Tutor:	Carlos A. Iglesias Fernández
Departamento:	Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	Mercedes Garijo Ayestarán
Vocal:	Álvaro Carrera Barroso
Secretario:	Juan Fernando Sánchez Rada
Suplente:	Tomás Robles Valladares

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO FIN DE GRADO

DESIGN AND IMPLEMENTATION OF A SEMANTIC TASK AUTOMATION SYSTEM FOR AGILE SOFTWARE PROJECT ENVIRONMENTS

Javier Díez Martínez

Julio de 2016

Resumen

Hoy en día, las metodologías ágiles son las más utilizadas para proyectos de desarrollo software. La metodología ágil es una alternativa a la gestión de proyectos tradicional, que ayuda a que los equipos respondan a los cambios de requisitos de forma continua mediante ciclos de trabajo iterativos, llamados sprints. Estas metodologías necesitan actualizaciones continuas sobre el estado del proyecto y de la realización de sus tareas. Para lograr esto, es muy útil tener una plataforma en la que se dé toda la información sobre el proyecto. Es tedioso para un desarrollador software actualizar esta información manualmente, por lo que su automatización tiene un gran valor.

El objetivo de este proyecto es el diseño e implementación de un sistema de automatización basado en tecnologías semánticas para entornos de proyectos de software ágiles. El proyecto define e implementa la arquitectura de un sistema que automatiza tareas que se realizan en el desarrollo ágil de software, basado en una plataforma de automatización de tareas mediante tecnologías semánticas. Además, ofrece al usuario una plataforma en la que se muestra la información del proyecto software proveniente de diferentes fuentes y proporciona un entorno para la organización del proyecto. Con el fin de lograr este objetivo, se ha adaptado una plataforma de automatización semántica de tareas y también se ha desarrollado una plataforma colaborativa que muestra la información del proyecto. La plataforma de información del proyecto se compone de varios submódulos que se conectan a la plataforma de automatización semántica. Estos actualizan automáticamente la información mostrada sobre el proyecto. La plataforma de automatización de tareas semántica se compone de varios submódulos que evalúan las reglas, que proporcionan la gestión de reglas y canales, y un módulo que dispara las acciones como resultado de la evaluación de las reglas. Este trabajo fin de grado ha desarrollado nuevos canales y reglas y también ha extendido la captura de eventos y el disparo de acciones. El sistema se ha validado en un caso de automatización de tareas en un entorno de un proyecto Scrum.

Por último, los problemas que se plantean durante el desarrollo, las conclusiones extraídas de este proyecto y las posibles líneas de trabajo futuro se exponen.

Palabras clave: Automatización, Scrum, Semántica, RDF, Desarrollo software, EWE

Abstract

Nowadays, agile methodologies are the most used for software development projects. Agile methodologies are an alternative to traditional project management tecniques, which help teams respond to continuous requirement's changes by iterative work cadences, known as sprints. These methodologies need of continuous updates on the project status at a task level (task done or pending). In order to achieve this, a platform that integrates all the software artefacts is desirable. It is tedious for a software developer to update manually this information in every tool, so the automation of this process would be extremely valuable.

The objective of this final project thesis is the design and implementation of a semantic task automation system for agile software project environments. The project defines and implements the architecture of a system which automates the tasks in an agile software development environment. To this end, the project proposes the use of a semantic task automation platform. Moreover, the project has developed a platform in which the software project's information from different sources is shown and provides a development environment for software organizations. In order to achieve this goal, the project aims at i) adapting the semantic task automation platform and ii) develop a software platform which integrates project's information and sets a collaborative organization framework.

The project information platform is composed of several submodules that are connected to the semantic automation platform. Based on these automations, the platform will provide updated information about the project. The semantic task automation platform is composed of several submodules that evaluate automation rules. Between these modules, the most relevant ones are rule and channel management and a module that triggers the actions as a result of evaluating the rules. In order to adapt this platform to its new use, new channels and rules has been created. In addition, new mechanisms for integrating events and triggering actions have been developed. The system developed in this project has been integrated in a real case, automating tasks in an agile software project development environment. Finally, the problems faced during the development, the conclusions drawn of this project and the possible lines of future work are exposed.

Keywords: Scrum, Automation, Semantics, RDF, EYE, EWE, Software Development

Contents

Resumen **V**

Abstract **VII**

Contents **IX**

List of Figures **XIII**

1 Introduction **1**

1.1 Motivation 1

1.2 Project goals 3

1.3 Structure of this document 3

2 Enabling Technologies **5**

2.1 Scrum 5

2.1.1 Scrum Roles 6

2.2 Docker 7

2.3 Github API 8

2.4 Ewe-tasker 9

2.4.1 Notation3 10

2.4.2 EYE 11

2.4.3 EWE Ontology 12

2.4.3.1 Channel 13

2.4.3.2 Event 13

2.4.3.3 Action 13

2.4.3.4 Rule 13

2.4.4 Channels In Ewe-tasker 13

2.4.4.1 Beacons 14

2.4.5 MongoDB 14

2.5 Restyaboard 14

3 Architecture **17**

3.1 Overview 17

3.2 Ewe Agile Board 19

3.2.1 GitHub Board 20

3.2.1.1 GitHub Integration 21

3.2.2 Scrum Board 23

3.2.3 Ewe Rules 24

3.3 Task Automation Server 25

3.3.1 Rule Engine 25

3.3.2 Rule Administration 26

3.3.3 Channel Administration 29

3.3.3.1 Events Manager 32

3.3.4 Action Trigger 32

3.3.4.1 Ewe Agile Board (EAB) integration 32

3.4 Mobile App 33

4 Case study **35**

4.1 Ewe Board configuration 35

4.2 Ewe tasker configuration 37

4.3 Case Study 39

5 Conclusions **41**

5.1 Conclusions 41

5.2 Achieved goals 42

5.3 Problems faced 43

5.4 Future work 44

Bibliography **45**

A New channels created **47**

List of Figures

1.1 Scrum Introduction 2

2.1 Scrum Framework 6

2.2 Docker 7

2.3 Docker Structure 8

2.4 GitHub 8

2.5 Ewe-tasker homepage 10

2.6 EYE integration 12

2.7 Evented WEb Ontology (EWE) Class Diagram 12

2.8 Kanban Board 15

3.1 Architecture 18

3.2 EAB Sub-modules Connections 19

3.3 GitHub Board Submodule 20

3.4 Oauth Overview 22

3.5 Scrum Board Submodule 24

3.6 TAS Sub-modules Interconnection 25

3.7 Rule Editor Interface 26

4.1 Register EWE Board 36

4.2 GitHub Board 36

4.3 Scrum Board 37

4.4 Ewe Rules Board 38

4.5 User Page Ewe tasker 38

4.6 Rule Import 39

A.1 EWE Board Channel 47

A.2 GitHub Channel 50

CHAPTER 1

Introduction

In this chapter, there is information about what motivates this project, what the project's main goals are, and finally a brief description of this document and its chapters.

Motivation

Nowadays, agile methodologies [1] are the most used for software development projects. Agile methodologies are an alternative to traditional project management techniques, which help teams respond to continuous requirement's changes by iterative work cadences, known as sprints.

Scrum is [2] the most popular agile methodology for software projects due to its simplicity and flexibility. Scrum emphasizes empirical feedback, team self management, and striving to build properly tested product increments within short iterations.

This methodology needs of continuous updates on a project status and on the tasks done or to do. In order to achieve this, a platform in which all the information is given is very important. It is tedious for a software developer to update this information, so the automation of this process would be extremely valuable.

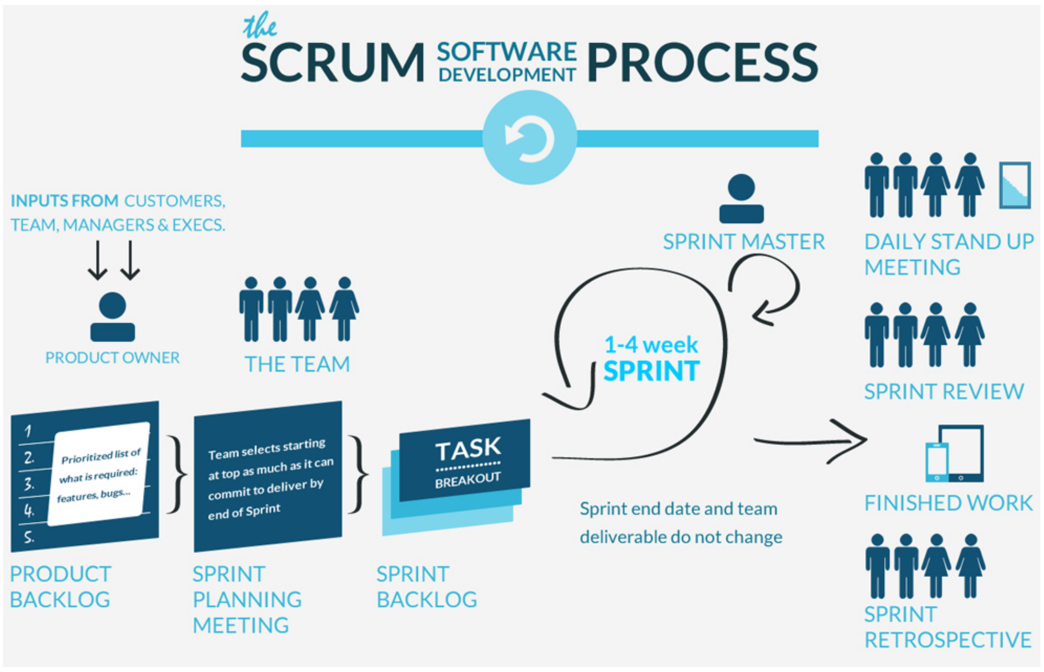

the SCRUM SOFTWARE DEVELOPMENT PROCESS
INPUTS FROM CUSTOMERS, TEAM, MANAGERS & EXECS.
PRODUCT OWNER
THE TEAM
SPRINT MASTER
DAILY STAND UP MEETING
1-4 week SPRINT
SPRINT REVIEW
Prioritized list of what is required: features, bugs...
Team selects starting at top as much as it can commit to deliver by end of Sprint
TASK BREAKOUT
FINISHED WORK
Sprint end date and team deliverable do not change
PRODUCT BACKLOG
SPRINT PLANNING MEETING
SPRINT BACKLOG
SPRINT RETROSPECTIVE

Figure 1.1: Scrum Introduction

A platform which automatically takes information out of the different tools that a software developer uses, and also sets a board for the software project administration, would be a very useful resource for its user.

Another motivation for the project would be to apply the smart home idea to the work place. The goal of smart homes is to make certain tasks easier for people, so this idea can be used for the work environment.

A smart home [3], or smart house, is a home that incorporates advanced automation systems to provide the inhabitants with sophisticated monitoring and control over the building's functions. For example a smart home may control lighting, temperature, multi-media, security, window and door operations, as well as many other functions.

The notion of smart home has evolved and has been adapted to workplaces which are known as *smart offices* [4]. In such smart offices, certain tasks related to a project could be automated, such as reminding of uploading the last version of a project when leaving.

In order to achieve the project's objectives, a task automation platform will be adapted for its use in a software project development, and also a web application that gives all information about a software project development will be developed.

Project goals

The main goal of this project is the adaptation of an intelligent automation platform based on ECA (Event-Condition-Action) rules to be used for agile software development methodologies, and the development of a platform which gives the user all the useful information about a project and facilitates agile methodologies principles to be applied.

This main goal includes some tasks such as:

- Develop a web application which gives all information about a software project and is updated automatically.
- Build a software module which is able to connect with the various web applications used by a software developer (Internet event sources).
- Connect the task automation server with the web application that gives project's information.
- Connect with action triggers in order to run actions generated by the rule engine.

Structure of this document

This document reflects how the design and implementation of a semantic task automation in agile software project environments has been done, why this project has been developed and conclusions that came up from the process. In this section we provide a brief description of each chapter included in this document. The structure is the following:

Chapter 1. Introduction. This chapter provides an introduction for the project, here both the motivation for the project's development and its main goals are detailed.

Chapter 2. Enable Technologies. This chapter gives a description of the main standards and technologies on which this project rely on.

Chapter 3. Architecture. This chapter explains the project's architecture and all the components and modules of the system.

Chapter 4. Case Study. This chapter offers an overview of the main use case. The running of the modules and their functionalities are explained, and the steps that a user has to follow to use this system.

Chapter 5. Conclusions. This chapter sums up the conclusions drawn from this project,and also explains problems faced and suggestions for future work.

CHAPTER 2

Enabling Technologies

In this chapter, the enabling technologies are introduced. Enabling technologies are the various already functional platforms and software that are used in order to implement the automation of tasks in an agile software project environment. In this chapter the agile methodology in which this project is based, Scrum, will also be explained.

Scrum

Scrum [5] is an Agile framework for completing complex projects. Scrum originally was formalized for software development projects, but it works well for any complex, innovative scope of work. The possibilities are endless. The Scrum framework is deceptively simple.

The Scrum Framework explained:

- A product owner creates a prioritized wish list called a product backlog.
- During sprint planning, the team pulls a small chunk from the top of that wish list, a sprint backlog, and decides how to implement those pieces.
- The team has a certain amount of time — a sprint (usually two to four weeks) — to complete its work, but it meets each day to assess its progress (daily Scrum).

- Along the way, the ScrumMaster keeps the team focused on its goal.
- At the end of the sprint, the work should be potentially shippable: ready to hand to a customer, put on a store shelf, or show to a stakeholder.
- The sprint ends with a sprint review and retrospective.
- As the next sprint begins, the team chooses another chunk of the product backlog and begins working again.

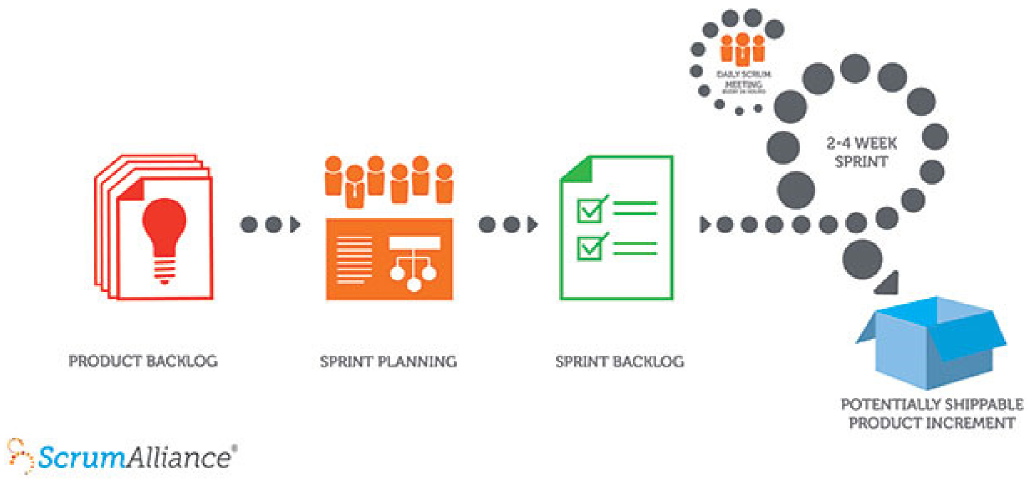


Figure 2.1: Scrum Framework

The cycle repeats until enough items in the product backlog have been completed, the budget is depleted, or a deadline arrives. Which of these milestones marks the end of the work is entirely specific to the project. No matter which impetus stops work, Scrum ensures that the most valuable work has been completed when the project ends.

Scrum Roles

- **The Product Owner** The Product Owner is responsible for maximizing the value of the product and the work of the Development Team. How this is done may vary widely across organizations, Scrum Teams, and individuals.
- **The Scrum Master** The Scrum Master is responsible for ensuring Scrum is understood and enacted. Scrum Masters do this by ensuring that the Scrum Team adheres to Scrum theory, practices, and rules. The Scrum Master is a servant-leader for the Scrum Team. The Scrum Master helps those outside the Scrum Team understand which of their interactions with the Scrum Team are helpful and which are not. The

Scrum Master helps everyone change these interactions to maximize the value created by the Scrum Team.

- **The Team Member** professional who does the work of delivering a potentially releasable Increment of “Done” product at the end of each Sprint.

Docker

Docker [6] is an open-source project that automates the deployment of applications inside software containers. Docker containers wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries – anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment.

Figure 2.2: Docker

These containers are used to run the task automation platform and the project information web application in a server, so that they are both accessible for users.

The reasons for using docker containers are the following:

- Lightweight. Containers running on a single machine share the same operating system kernel; they start instantly and use less RAM. Images are constructed from layered file-systems and share common files, making disk usage and image downloads much more efficient.
- Open. Docker containers are based on open standards, enabling containers to run on all major Linux distributions and on Microsoft Windows – and on top of any infrastructure.
- Secure. Containers isolate applications from one another and the underlying infrastructure, while providing an added layer of protection for the application.

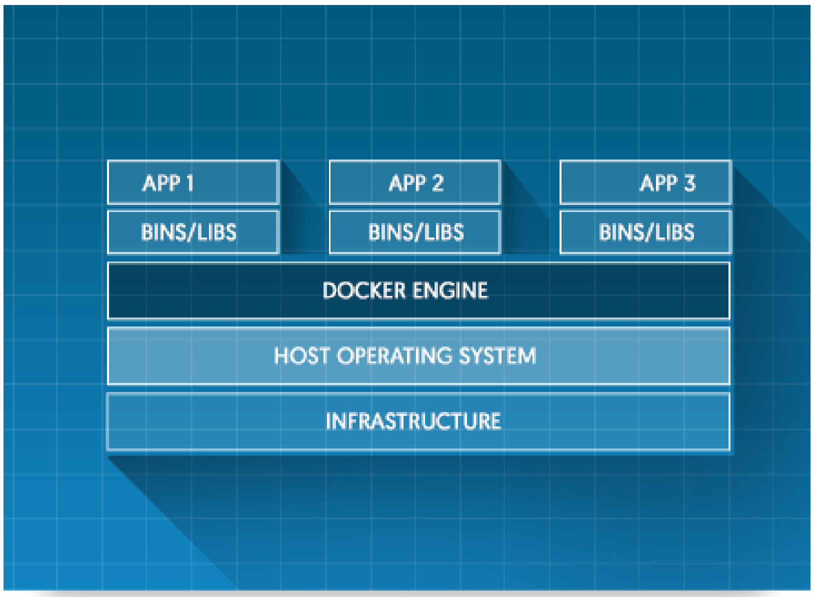


Figure 2.3: Docker Structure

Github API

GitHub [7] is a web-based Git repository hosting service. It offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features. Unlike Git, which is strictly a command-line tool, GitHub provides a Web-based graphical interface. It also provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.[1]

Figure 2.4: GitHub

[1]https://en.wikipedia.org/wiki/GitHub

An application programming interface (API) is a particular set of rules ('code') and specifications that software programs can follow to communicate with each other. It serves as an interface between different software programs and facilitates their interaction, similar to the way the user interface facilitates interaction between humans and computers.[2]

Connection with GitHub is critical for this project because it is a common tool used by software developers from which a lot of information about a software project can be obtained. For this project the API that GitHub has created for developers will be used, so that a connection between platforms can be achieved easily.

All GitHub API access is over HTTPS, and all data is sent and received as JSON. Methods documented by GitHub for this API will be used in this project's development.

Ewe-tasker

Ewe-tasker [8] is a web application which does rule-based task automation. Rules used by this platform have an Event-Condition-Action structure, so that when an event takes place the conditions are evaluated, and if conditions are matched then the action is executed. These rules are written in Notation3.

Ewe-tasker provides these services:

- Users can create rules for a specific task automation.
- Users can create channels that have events and actions. Channels are used when creating rules.
- Users can test rules with the EYE(Euler YAP Engine) reasoner.
- Users can import existing rules for their own use.

In this project, Ewe-tasker will be modified in order to adapt it to a software development project environment which uses Scrum methodology.

In the next sections the several technologies that enable Ewe-tasker will be explained.

[2]http://stackoverflow.com/questions/7440379/what-exactly-is-the-meaning-of-an-api

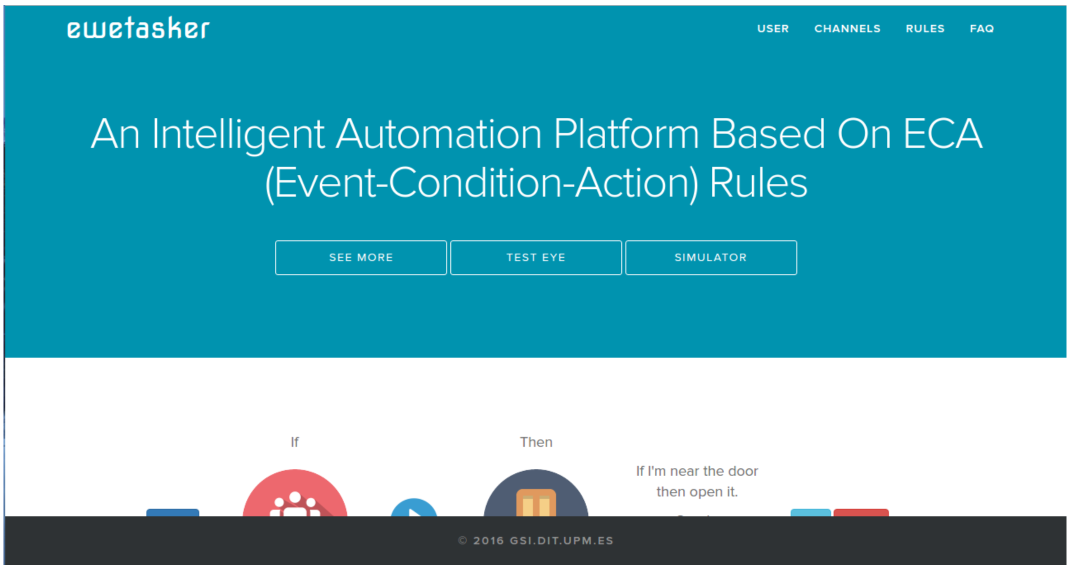

ewetasker
USER
CHANNELS
RULES
FAQ
An Intelligent Automation Platform Based On ECA (Event-Condition-Action) Rules
SEE MORE
TEST EYE
SIMULATOR
If
Then
If I'm near the door then open it.
© 2016 GSI.DIT.UPM.ES

Figure 2.5: Ewe-tasker homepage

Notation3

Notation3 [9], or N3 as it is more commonly known, is a shorthand non-XML serialization of Resource Description Framework models, designed with human-readability in mind: N3 is much more compact and readable than XML RDF notation. The format is being developed by Tim Berners-Lee and others from the Semantic Web community. A formalization of the logic underlying N3 was published by Berners-Lee and others in 2008. N3 has several features that go beyond a serialization for RDF models, such as support for RDF-based rules.[3]

The aims of N3 are:

- To optimize expression of data and logic in the same language.
- To allow RDF to be expressed.
- To allow rules to be integrated smoothly with RDF.
- To allow quoting so that statements about statements can be made.
- To be as readable, natural, and symmetrical as possible.

The language achieves these objectives with the following features:

[3]https://en.wikipedia.org/wiki/Notation3

- URI abbreviation using prefixes which are bound to a namespace (using @prefix) a bit like in XML.
- Repetition of another object for the same subject and predicate using a comma ",".
- Repetition of another predicate for the same subject using a semicolon ";".
- Bnode syntax with a certain properties just put the properties between [and].
- Formulae allowing N3 graphs to be quoted within N3 graphs using { and }.
- Variables and quantification to allow rules, etc to be expressed.
- A simple and consistent grammar.

N3 [10] differentiates itself from other rule languages because of its expressiveness. For example, in N3 it is possible to create rules in the consequence, and to use built-ins. The N3 logic has monotonicity of entailment, which means that the hypotheses of any derived fact may be freely extended with additional assumptions, which is an important property when reasoning about a changing knowledge base [11]. Some engines like FuXi7 or EYE support N3 syntax, but EYE reasoner is the only which support all N3's expressions.

EYE

The EYE (Euler YAP Engine) [12] reasoner is a high-performance reasoning engine that uses an optimized resolution principle, supporting forward and backward reasoning and Euler path detection to avoid loops in an inference graph. It is written in Prolog and supports, among others, all built-in predicates defined in the Prolog ISO standard. Backward reasoning with new variables in the head of a rule and list predicates are a useful plus when dealing with OWL[4] ontologies, so is more expressive than RDFox or FuXi, whilst being more performant than other N3 reasoners.

EYE translates Notation3 to Prolog Coherent Logic code and runs it on YAP[5] engine. This engine supports monotonic abduction-deduction-induction reasoning cycle. EYE can be configured with many options of reasoning and can also provide useful information of reasoning.

The engine can be added multiple features by new user-defined plugins.

[4]Web Ontology Language: the ontology description standard as defined by the World Wide Web Consortium (W3C)

[5]YAP (Yet Another Prolog)

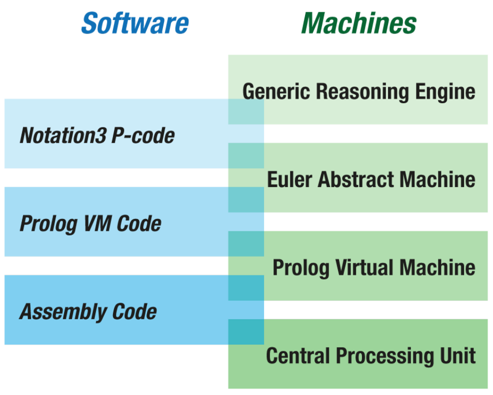


Figure 2.6: EYE integration

EWE Ontology

EWE [13] is a standardized data schema (also referred as ”ontology” or ”vocabulary”) designed to describe elements within Task Automation Services enabling rule interoperability. Referring to the EWE definition [14].

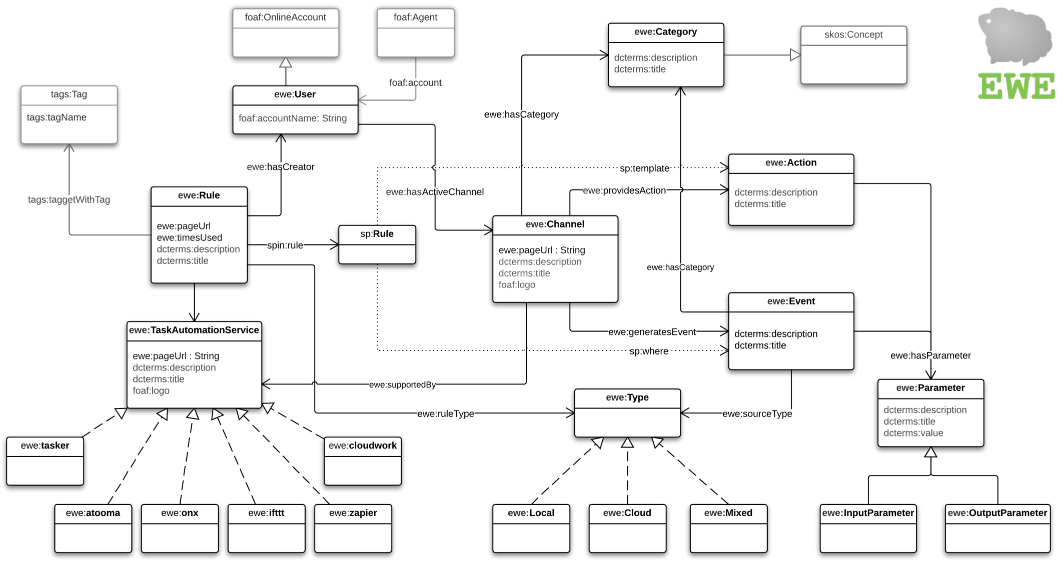


Figure 2.7: EWE Class Diagram

The ontology has four main classes: Channel, Event, Action and Rule.

Channel

It defines individuals that either generate Events, provide Actions or both. In the context we refer, Channel mostly defines Web Services. However, sensors and actuators are also described as channels, thus they produce events or provide actions (e.g. a GPS device programmed to generate alerts when it is near certain locations).

Event

Event class defines a particular occurrence of a process, they are generated by a particular Service (e.g. new-chat-message events are generated by GoogleTalk service). Events have no duration. Changes on the state of a system or a sensor, can be modeled as events (the change in the state triggers the Event generation).

Action

Action class defines an operation or process provided by a Service. Actions produce effects whose nature depend on the action's nature (e.g. producing a log message, modifying a public or private state on a server, switching on a light, etc.). Actions are on the right part of the Rule. Web services, actuators, and smartphone apps can provide actions when modeled as Services.

Rule

Rule defines an ECA rule, triggered by an Event that means the execution of an Action. It defines particular interconnections among instances of event and action, that includes the configuration parameters set for both of them. Rules are defined as a SPARQL construct query by means of the property spin:rule, using the SPIN language.

Channels In Ewe-tasker

These are a number of channels available in Ewe-tasker:

- Presence Sensor: This channel represents a presence sensor.
- Smart TV: This channel represents a simplified Smart TV with simple capabilities.
- Toast Notification: This channel represents an android toast notification.

- Notification: This channel represents a smartphone notification.
- Twitter: This channel represents Twitter social network.
- Connected Door: This channel represents a connected door lock able to detect when the door is opened, closed or shut, but it also can open, lock or unlock the door.

Beacons

Beacons [8] are small battery devices able to emit a Bluetooth Low Energy (BLE) signal that can be captured by smartphones situated inside its broadcast range devices. This signal is often relayed to a cloud server that processes the information and treats it according to its nature. Information travels by this signal and can be anything from environment data (temperature, air pressure, humidity) to indoor positioning data (asset tracking, retail) or orientation (acceleration, rotation).

MongoDB

MongoDB[6] [15] is an open source database that uses a document-oriented data model. Instead of using tables and rows as in relational databases, MongoDB is built on an architecture of collections and documents. Documents comprise sets of key-value pairs and are the basic unit of data in this database. Collections contain sets of documents and function as the equivalent of relational database tables.

Like other NoSQL databases, MongoDB supports dynamic schema design, allowing the documents in a collection to have different fields and structures. The database uses a document storage and data interchange format called BSON, which provides a binary representation of JSON-like documents. Automatic sharding enables data in a collection to be distributed across multiple systems for horizontal scalability as data volumes increase.

MongoDB provides high performance, high availability, and easy scalability.

Restyaboard

Restyaboard [16] is an online kanban board based on Restya platform.

A Kanban board is a work and workflow visualization tool that enables you to optimize the flow of your work. Physical Kanban boards, like the one pictured below, typically use

[6]http://www.mongodb.org/

sticky notes on a whiteboard to communicate status, progress, and issues. Online Kanban boards draw upon the whiteboard metaphor in a software setting.

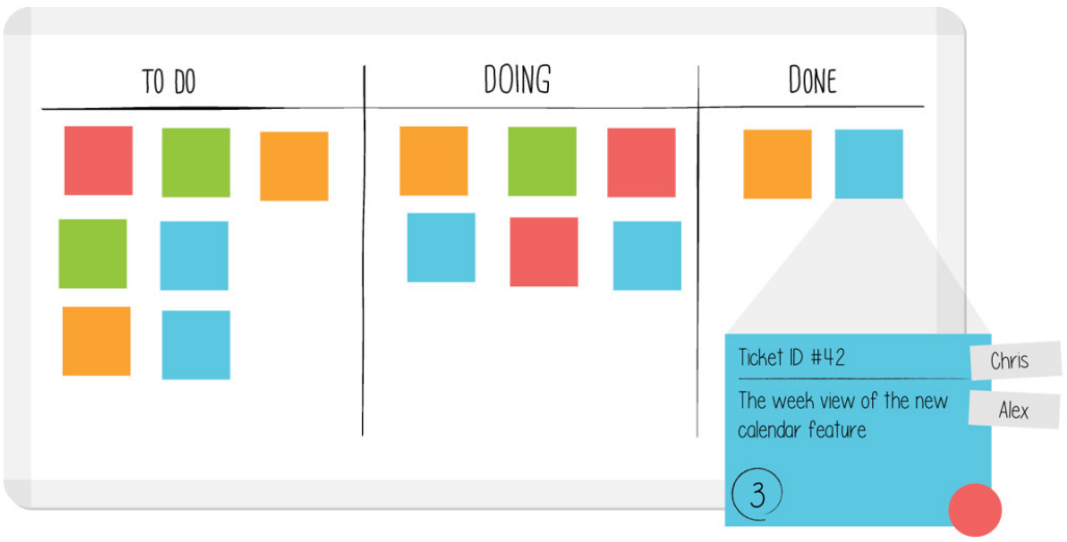


Figure 2.8: Kanban Board

Restya platform is tech agnostic and so available in Java, Node.js, Python, PHP, .Net variants. It is based on REST and SOA (Service Oriented Architecture) and designed for scalabilty, mobile friendly, cloud friendly.

This platform is the base for the EAB, its code is modified in order to achieve the final project's goals. Restyaboard provides of a base for this project, which already has some important features such as an API Rest.

CHAPTER 3

Architecture

Overview

In this chapter, the architecture of this project will be explained, both design and implementation phases details. The system is divided into several modules to help its understanding. First the global system will be explained and after that, each module details will be given.

The figure 3.1 shows the whole system, which is composed by two modules described below. Each module is divided into several submodules.

- **Ewe Agile Board (EAB)**: this module aims to show the information about a project. It listens to events from GitHub repositories issues and also shows the rules available in the Task Automation Server (TAS). This module is divided into three submodules:
  - Github
  - Scrum Board
  - Ewe Rules
- **Task Automation Server (TAS)**: this module gives the user the functionality of

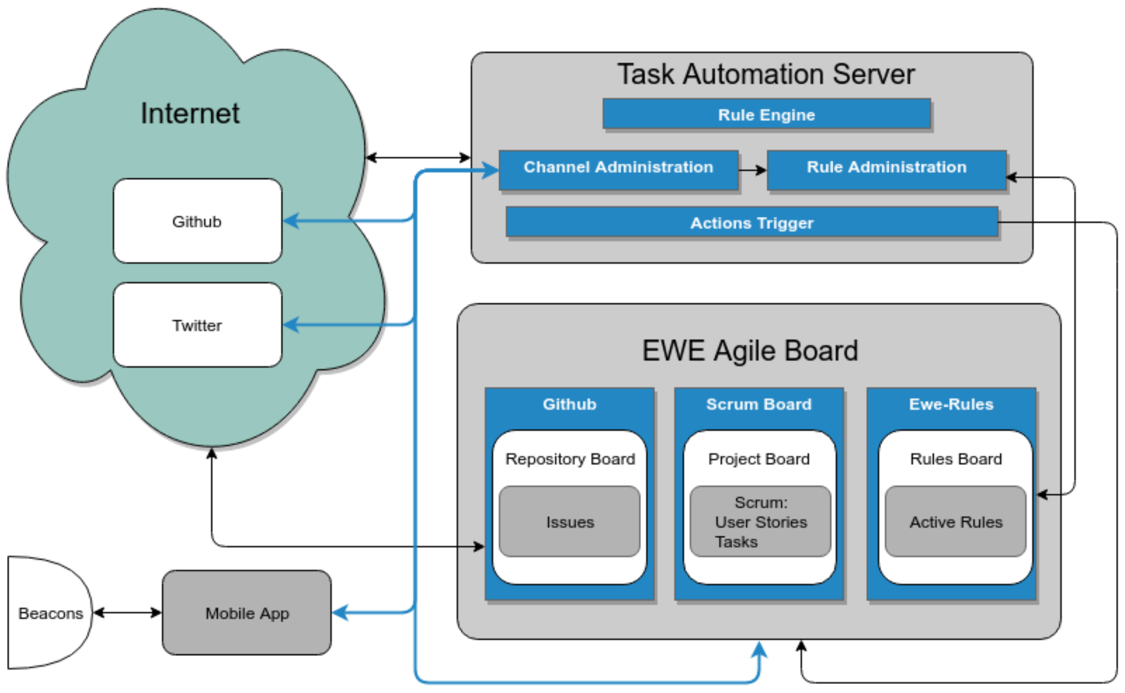


Figure 3.1: Architecture

managing the rules and creating new ones. Its main role is to receive events from different sources, such as GitHub, and to trigger actions established in the rules.. This module is divided into four sub-modules:

- Rule Engine
- Rule Administration
- Channel Administration
- Action Trigger

Events coming from Beacons are passed to the TAS via the Mobile App. Other events come from different sources in the Internet, such as GitHub, directly to the TAS. Once the events are received an action is triggered, so the TAS sends a command to the application that has to do the action. In this case, EAB receives the command and executes the action.

This project is mainly focused on the EAB and also in a minor way the TAS.

Ewe Agile Board

The main purpose of this project is to give a platform where you can obtain all the information related to a software development project, and that this information is easily accessible and automatically updated. It is really important to have all this information if an agile methodology is used for the development, because it facilitates a better and overall more consistent organization of the project. This information is given in the several submodules of EAB and it is automatically updated through the TAS. Each submodule shows different relevant information the **GitHub Board** shows information related to the project's GitHub repository, the **Ewe Rules** shows the several rules available in the TAS and the **Scrum Board** shows the Scrum methodology information related to the project, such as the Sprint Backlog.

The connection between these submodules does not exist, it is the duty of the Product Owner and also in a minor way of the Team Members to work in the **Scrum Board** taking into account the information given in the **GitHub Board**. There is a connection between GitHub and the **GitHub Board** and also a connection between the TAS and the **Ewe Rules** submodule. This is shown in the Figure 3.2.

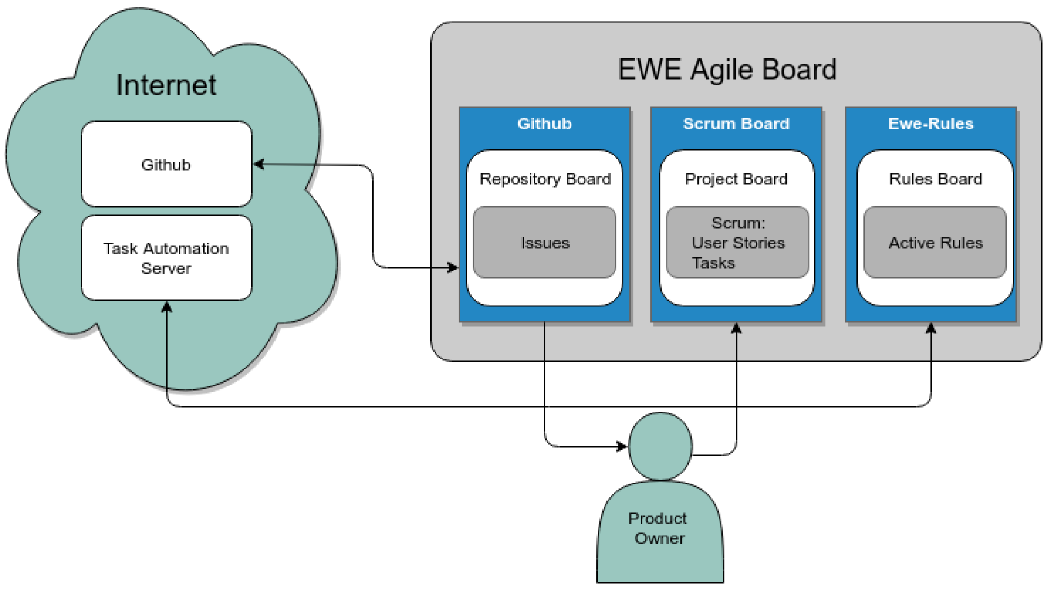


Figure 3.2: EAB Sub-modules Connections

GitHub Board

This is one of the most important modules of this project, it receives information related to the GitHub repository for the project, and it shows it to the user in a board view with lists and cards. The information needed for this board is all obtained automatically through GitHub WebHooks, although it can come directly from GitHub to update the repository issues status or from the TAS, if it is information related to the repository members or commits, as shown in Figure 3.3.

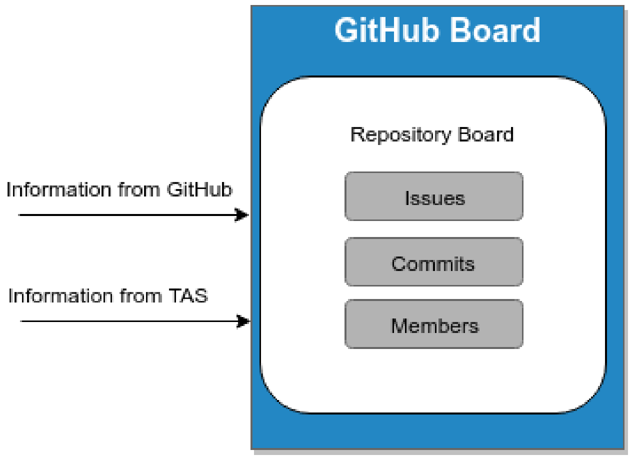


Figure 3.3: GitHub Board Submodule

All information is received via HTTP Posts. These Posts generate new Cards or update an already existing card. It is very important to give the user a view where it is easy to find the information, so it does not make sense to give a lot of details in each card. Every GitHub repository issue has its own card and it can be, depending on its status in one of these lists:

- New: when an issue has just been posted, without assigned user, comments or milestone date.
- Assigned: when an issue has been assigned to a user.
- In Progress: when an issue has milestone date.
- Closed: when an issue is not open.
- Feedback: when an issue has been commented.

There is another two lists, one where the repository new members are shown (every member has a card) and the other shows commits done to the repository(every commit has a card).

The GitHub Board can be shared by several users so that the whole development team can have access to the information.

GitHub Integration

In this section, how GitHub is integrated in the system will be explained.

In order to access GitHub an authorization is needed so the application must be registered in GitHub as a developer application. After registering the application can use the GitHub Oauth API. The application is registered with a callback public URL where all traffic from GitHub to the application will be send.

OAuth (Open Authorization) is an open standard for token-based authentication and authorization on the Internet.

OAuth allows an end user's account information to be used by third-party services, such as Facebook or GitHub, without exposing the user's password. OAuth acts as an intermediary on behalf of the end user, providing the service with an access token that authorizes specific account information to be shared. The process for obtaining the token is called a flow.[1]

Each EAB user that wants to use the GitHub Board module via the GitHub application has to log in to GitHub from the EAB, when the user logs in the EAB platform receives a token, this token is unique to its user and it can be used to execute different actions in GitHub.

The EAB platform saves the token for its use. All actions related to GitHub are done via the GitHub API. The desired action in order to achieve the issue information functionality is to subscribe to the user's repositories webhooks.

Webhooks are user-defined HTTP callbacks. They are usually triggered by some event, such as pushing code to a repository or a comment being posted to a blog. When that event occurs, the source site makes an HTTP request to the URL configured for the webhook.[2]

In our case GitHub Webhooks send the EAB the information of every action related to the user's GitHub repositories. This information is received in JSON format so it has to be appropriately processed in order to execute the correspondent action in the EAB platform.

[1]http://searchsoa.techtarget.com/

[2]https://en.wikipedia.org/wiki/Webhook

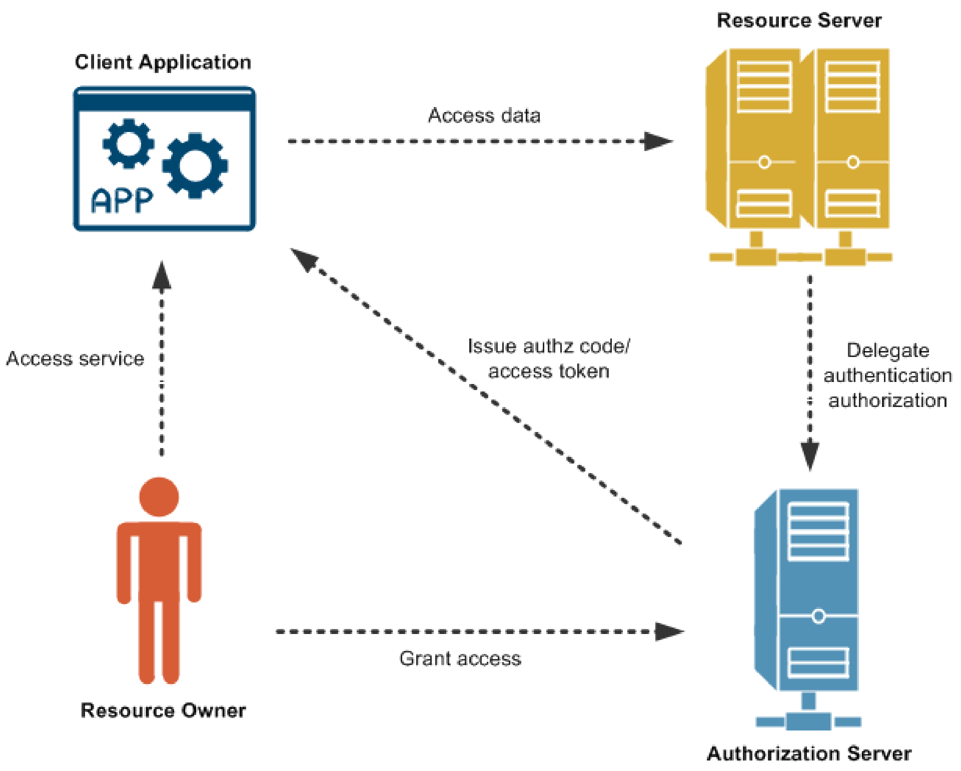


Figure 3.4: Oauth Overview

This is an example of the JSON received via GitHub Webhook when a member is added to a repository:

Listing 3.1: ”JSON GitHub Webhook”

```
{
  {
    "action": "added",
    "member": {
      "login": "octocat",
      [...]
    },
    "repository": {
      "id": 35129377,
      "name": "public-repo",
      "full_name": "baxterthehacker/public-repo",
      "owner": {
      "login": "baxterthehacker",
      "id": 6752317,
      [...]
    },
    "private": false,
    [...]
    "sender": {
      "login": "baxterthehacker",
      [...]
    }
  }
}
```

Only part of the JSON file is shown due to to its length. It can be appreciated that this JSON has very useful information about the event that has taken place. A new member was added to the repository called “public-repo” and the name of this new member is “octocat”.

This is valuable information that will be presented to the user in the GitHub Board so that the user can obtain information about the project’s GitHub repository in a very simple way.

Scrum Board

This is also one of the most important modules of this project, it gives the user a view where he can see all the information related to the project from a Scrum methodology point of view. This board can be shared with all the team members so that everyone can modify the information on it and the Scrum methodology can be used easily.

This module allows several users to collaborate in order to make the project organization clearer for all of them and also more dynamic, as shown in Figure 3.5.

The lists offered in this board are:

- Vision: the overall goal of the project.
- Release Planning: User Stories and when they have to be implemented and completed.
- Sprint Planning: structure of a Sprint.
- Sprint Backlog: tasks that have to be done during the Scrum Sprint.
- Sprint Execution: tasks in progress.
- Sprint Review: what has been accomplished during the Scrum Sprint.
- Sprint Retrospective: improvements for the development process.
- Sprint Release: the result of the Scrum Sprint.

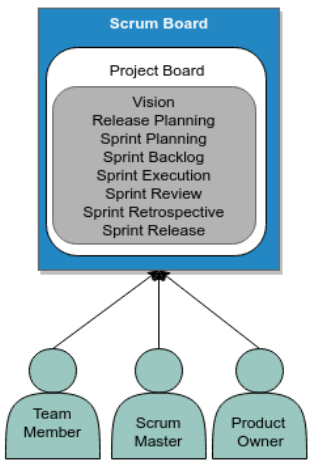


Figure 3.5: Scrum Board Submodule

Ewe Rules

This submodule shows the rules available in the TAS, which are shown in a board view that is automatically created for every new user. These rules are received via HTTP Post from the TAS.

The importance of this submodule is that the user gets to know that using the TAS is necessary in order to fully take advantage of the platform.

Task Automation Server

The main purpose of this module [8] in this project is to automate certain tasks in order to make a software development project easier for the developers team. This module is composed by 4 modules:

- Rule Engine: submodule that evaluates the rules that have been created. If the conditions are matched an action is triggered.
- Rules Administration: submodule that allows the user to create and manage the rules. These rules have events and actions supplied by the channels.
- Channel Administration: submodule that allows the user to create and manage the channels. It also manages events and sends the response with the action to the Action Trigger.
- Action Trigger: submodule that triggers an action when it receives the appropriate response.

The Figure 3.6 shows how these submodules are connected with each other:

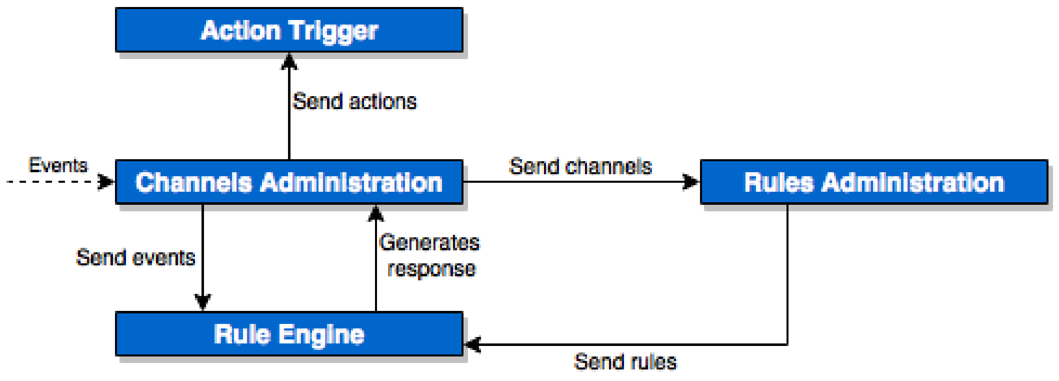


Figure 3.6: TAS Sub-modules Interconnection

Rule Engine

Rule Engine is a very important submodule, and it is based on the EWE ontology [17]. It is composed by the EYE Server and the EYE Helper.

The process of handling events and rules and triggering actions is the following:

- A new event is received.

- The EYE Helper captures it.
- The EYE Helper loads the rules from Rules Administration.
- The EYE Helper sends these events and rules to the EYE Server.
- The EYE Server evaluates the received data and generates a response.
- The EYE Helper sends the Notation3 response to the Channel Administration to be parsed.
- This response is parsed and the actions are triggered by the Action Trigger.

Rule Administration

This submodule is responsible for the rule creation and management. It is composed by Rule Editor, Rule Manager and Rule Repository.

The rule has an "If this then that" structure so the if the condition is matched the action is triggered. For example: "If an issue is posted in my project's GitHub repository then create a card in the project's EAB board".

Figure 3.7 shows the graphical interface for rule editing.

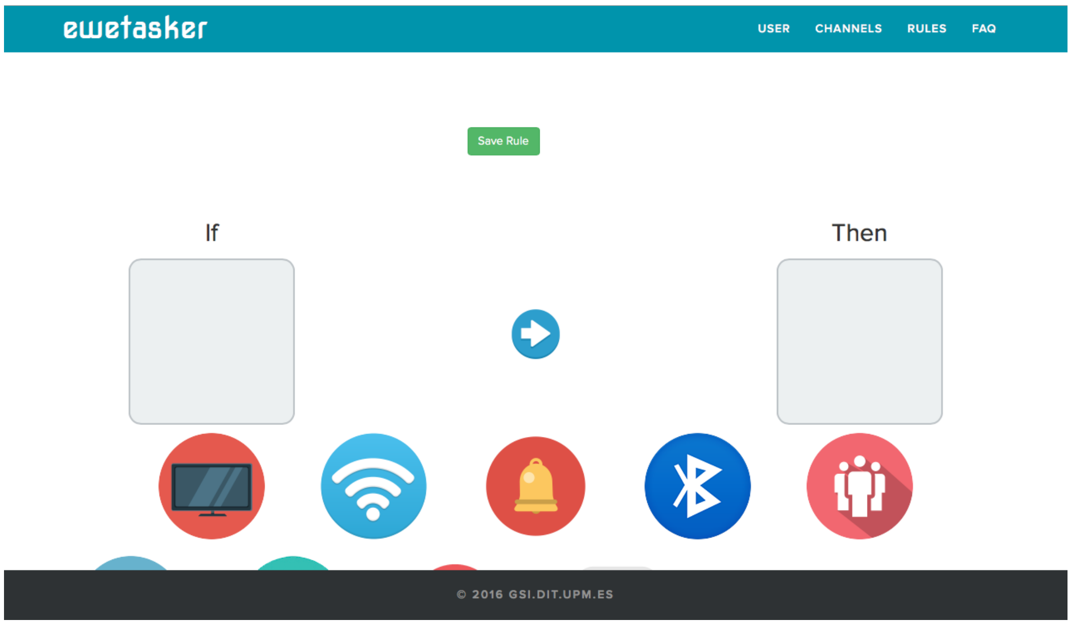


Figure 3.7: Rule Editor Interface

New rules created for this project, to automate tasks related to an agile software development project environment, are the following:

- If there is a new commit, a Commit card is created in the correspondent GitHub Board in EAB.
- If a new member is added to a GitHub repository, a Member card is created in the correspondent GitHub Board in EAB.
- If there is a new comment on a commit, this comment is added to the correspondent Commit card in EAB.
- If the user leaves the work place, the user is notified to commit any changes made to the project.

Rules are stored in the Rule Repository using JSON, as shown in Listing 3.2.

Listing 3.2: "Rule stored"

```
{
  "title": "New Commit Rule",
  "place": "GSI",
  "description": "If commit has been done, then switch create a new commit
      card.",
  "created_by": "Javier",
  "event_channel": "GitHub",
  "action_channel": "EWEBoard",
  "event":{
    "title": "New Commit",
    "prefix": "@prefix ewe-github: <http://gsi.dit.upm.es/ontologies/ewe-
        github/ns/#> .",
    "eye_fragment": "?event rdf:type ewe-github:newCommit."
  },
  "action":{
    "title": "Post Commit Card",
    "prefix": "@prefix ewe-board: <http://gsi.dit.upm.es/ontologies/ewe-
        board/ns/#> .",
    "eye_fragment": "ewe-board:EWEBoard rdf:type ewe-board:postCommitCard
        ."
  },
  "eye_rule":"
  {
    ?event rdf:type ewe-github:newCommit. # Received event of type
        newCommit
    ?event ewe:timeCommit ?timeCommit. # With a time.
    ?event ewe:authorCommit ?authorCommit. # With an author.
    ?event ewe:messageCommit ?messageCommit. # With a message.
    ?event ewe:idCommit ?idCommit. # With a commit ID.
    ?event ewe:URLCommit ?URLCommit. # With an URL.
  }
  =>
  {
    ewe-board:EWEBoard rdf:type ewe-board:postCommitCard ;
    ov:timeCommit ?timeCommit ; # With a time.
    ov:authorCommit ?authorCommit ; # With an author.
    ov:messageCommit ?messageCommit ; # With a message.
    ov:idCommit ?idCommit ; # With a commit ID.
    ov:URLCommit ?URLCommit . # With an URL.
  }.",
  "created_at":"16:22 14-07-2016"
}
```

It is crucial that we take all the important information from the event via the parameters and take it into consideration for the action.

Channel Administration

This module provides a channel editor so that new channel can be created and existing channels can be modified. It also is responsible for event handling and the parsing of the response generated when evaluating a rule so that it can send the Action Trigger what actions must be triggered.It is composed by: Channel Editor, Channel Manager, Channel Repository and Events Manager.

Channels are also stored in JSON in the Channel repository. Listing 3.3 shows a new channel created for this project, with just one of its events.

Listing 3.3: "New Channel "

```
{
  "title": "GitHub",
  "description": "This channel represents GitHub.",
  "nicename": "GitHub",
  "created_by": "Javier",
  "events": {
    "event": {
      "title": "New Commit",
      "prefix": "@prefix ewe-github: <http://gsi.dit.upm.es/ontologies/ewe-
          github/ns/#> .",
      "num_of_params": "5",
      "eye_fragment": "?event rdf:type ewe-github:newCommit. # Received
          event of type newCommit
        ?event ewe:timeCommit ?timeCommit. # With a time.
        ?event ewe:authorCommit ?authorCommit. # With an author.
        ?event ewe:messageCommit ?messageCommit. # With a message.
        ?event ewe:idCommit ?idCommit. # With a commit ID.
        ?event ewe:URLCommit ?URLCommit. # With an URL."
    }
    [...]
  },
  "actions:": {
    action":{
      "title": "",
      "prefix": "",
      "num_of_params": "",
      "eye_fragment": ""
   }
  },
  "created_at":"15:30 14-07-2016"
}
```

There are two completely new channels created for this project, below a list of their actions and events is shown with their respective parameters:

- GitHub Channel. Events:
  - New Commit.When a new push event to a repository takes place. Params:
    - Commit author
    - Commit message
    - Commit time

        - Commit repository
        - Commit ID
        - Commit URL
    - New Comment on a Commit.When someone comments a commit. Params:
        - Comment author
        - Comment message
        - Comment time
        - Commit repository
        - Commit ID
        - Commit URL
    - New Member.When a new member is added to a repository. Params:
        - Repository
        - New member

- EWE Board Channel. Actions:
    - Post Commit card.Creation of a new Commit card on a GitHub Board. Params:
        - Commit author
        - Commit message
        - Commit time
        - Commit repository
        - Commit ID
        - Commit URL
    - Post Commit Comment . A comment is added to a Commit card. Params:
        - Comment author
        - Comment message
        - Comment time
        - Commit repository
        - Commit ID
        - Commit URL
    - Post Member.Creation of a new Member card on a GitHub Board. Params:
        - Repository
        - New member

Events Manager

This module [8] is responsible for capturing events and parsing the response generated when a rule is evaluated so that it can send the Action Trigger the actions to be triggered.

New specific code has to be made in order to capture the events of new channels. When an event is captured the rule is evaluated and the response is parsed. After parsing, the result is send to the Action Trigger module.

The way the events coming from GitHub are captured is via Webhooks, as explained in Section 3.2.1.1. The JSON received is parsed in order to make it compatible for the rule engine.

Action Trigger

A JSON arrives with the response from the rule engine to this module. It is processed so that the right action is triggered. An example of an action would be posting a new card in a EAB board. The Action Trigger sends an HTTP post to the EAB API to execute the action, and finally a card is created as desired. The JSON response structure is shown in Listing 3.4.

Listing 3.4: "Response"

```
{
  "channel":"Notification",
  "action":"remindCommit",
  "parameter":"Don’t forget to commit any changes made to the project!"
}
```

EAB integration

Actions from the EWE Board channel must be executed, in order to do this, the Action Trigger module uses the API os the EAB platform. Action Trigger module sends HTTP requests to the EAB platform with the proper information as documented in [16][3].

HTTP requests are received in the EAB platform where finally the desired action is executed, whether it is creating a new Commit card or adding a new comment.

[3]http://board.demo.restya.com/api-explorer/

Mobile App

A Mobile App [18] is used in this project so that the beacons and rules with them can be used. The mobile receives from each beacon via Bluetooth data and processes it, so that an event is captured and a rule is evaluated.

In this chapter the modules that form this project's architecture have been explained, along with their connection and relationship.

In conclusion, this project has one main module, which is the EAB, but also has another important module which is the TAS. The main module (EAB) provides the user a platform in which a project's information is shown and supports itself on the TAS to update that information automatically, and provide more advantages to the user.

In the following chapter, the case study is explained in detail.

CHAPTER 4

Case study

In this chapter the main use case will be described so that it helps to a better understanding of the system functionalities by showing the main features of the system.

The main actor of this case is the user, whose goal is to configure the various platforms so that the automated environment is set. Following the steps described below, users will be able to take full advantage of the system.

Ewe Board configuration

First step in order to use the EAB is to create and verify an account. The user will have to register with a user name, an email and a password. The user has to fill the form shown bellow.

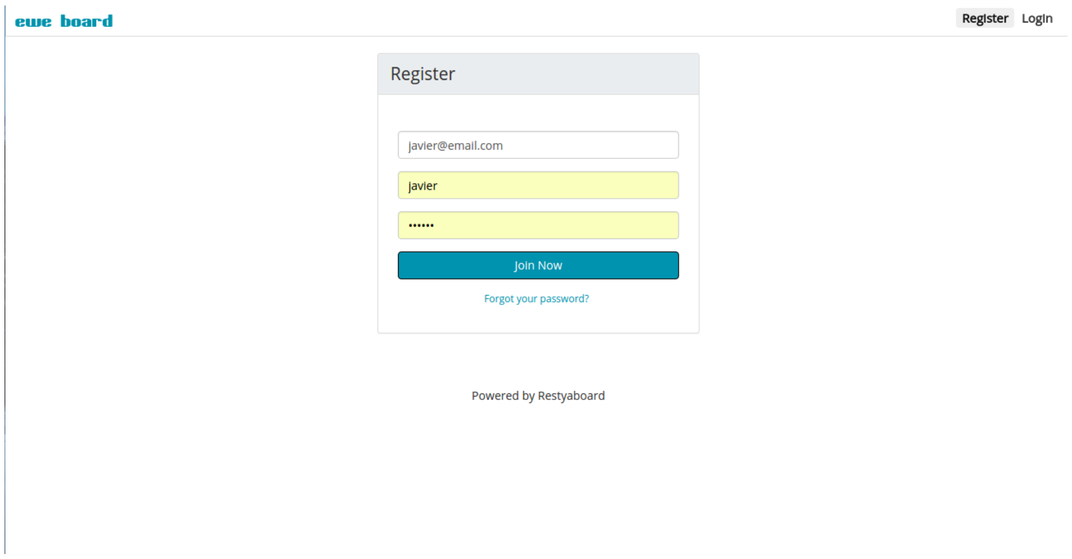


Figure 4.1: Register EWE Board

After registering, the user will receive an email from the platform giving an URL for the account activation. When the user accesses that URL the account is activated and the user is allowed to use the platform.

The user then must activate the GitHub App in the platform which will create a board for each repository of the GitHub account that the user has logged in to. This app can be accessed from the app menu in the page's footer, the option required is "Import from GitHub". The board created would have the following appearance shown in Figure 4.2.

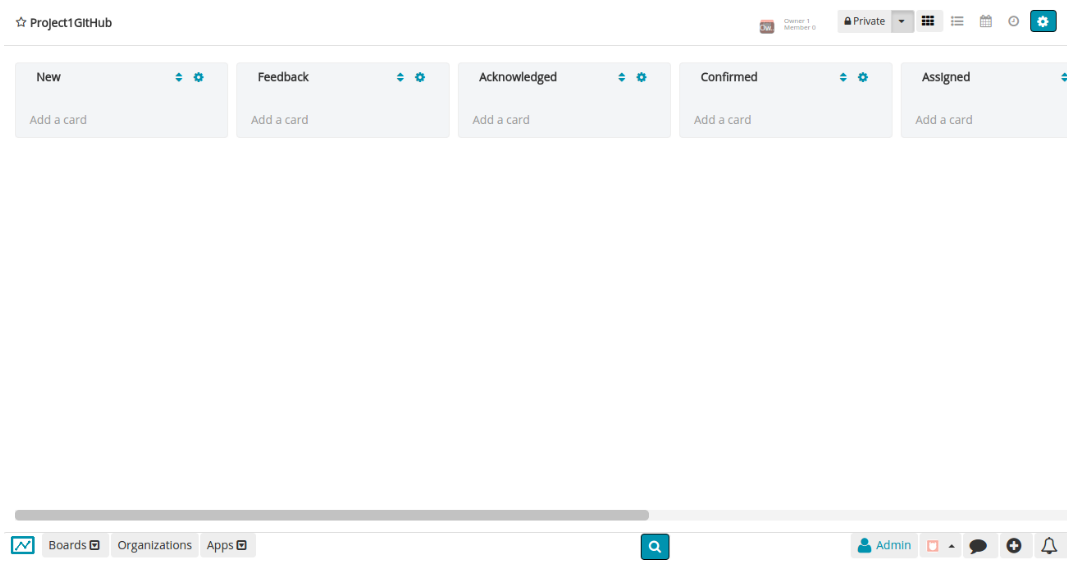


Figure 4.2: GitHub Board

A new board must be created for each project, this board is added by the user using the Scrum Template given by the platform. This board is created using the board menu in the page's footer, and its appearance is shown in Figure 4.3.

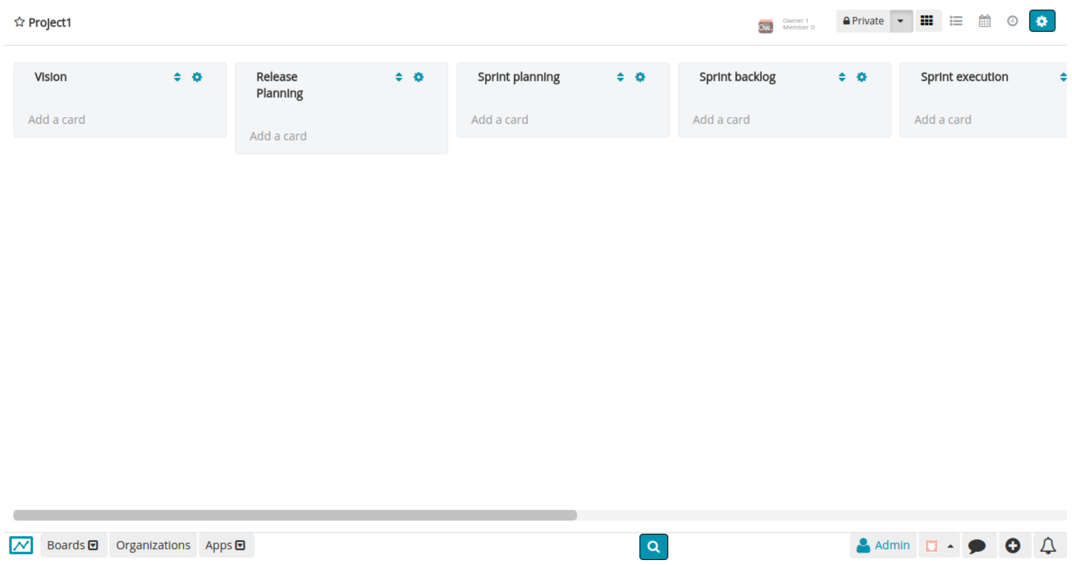
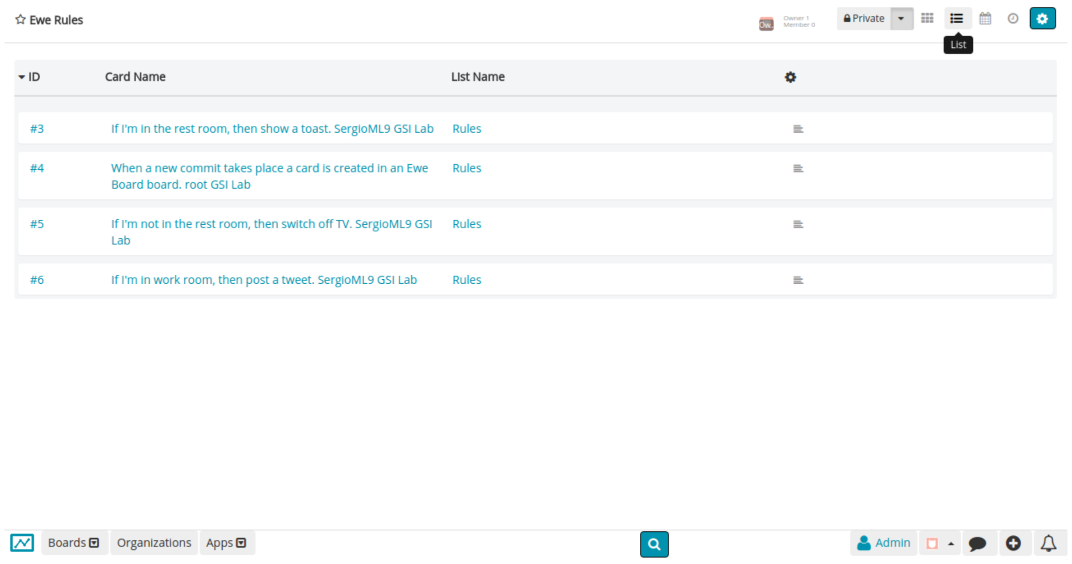


Figure 4.3: Scrum Board

Ewe tasker configuration

In order to use TAS Ewe tasker, users must create a new user account previously. This account creation does not require activation, but the user must then, from the user page, log in to the various platforms that have channels, so that the platform's channels are correctly configured for each user.

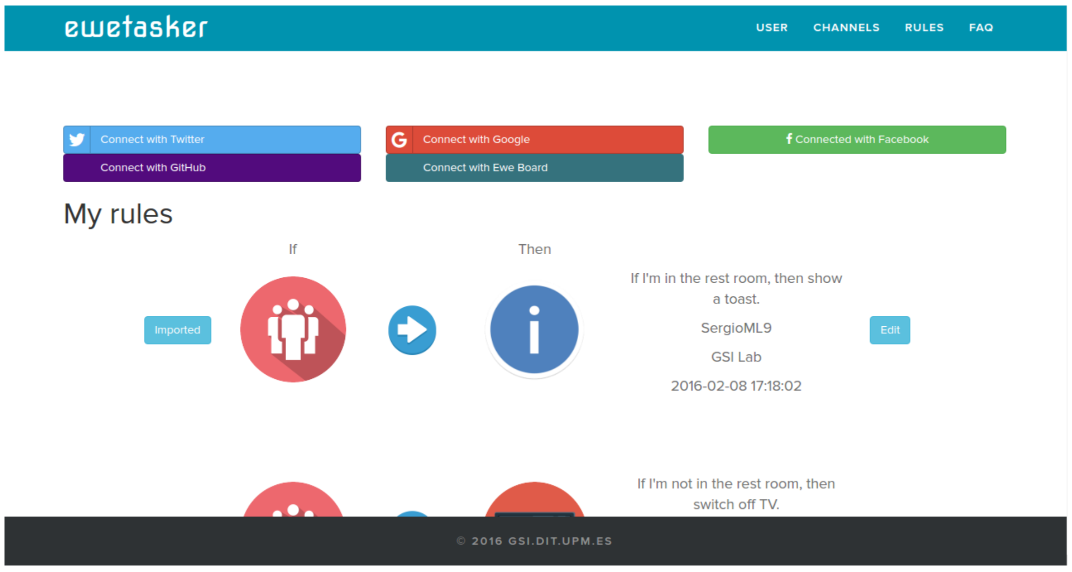
When logging into EAB from the TAS, a new board will be created in the EAB platform, its name is "Ewe Rules" and it shows every rule created in the TAS. Its appearance is shown in Figure 4.4.



ID	Card Name	List Name
#3	If I'm in the rest room, then show a toast. SergioML9 GSI Lab	Rules
#4	When a new commit takes place a card is created in an Ewe Board board. root GSI Lab	Rules
#5	If I'm not in the rest room, then switch off TV. SergioML9 GSI Lab	Rules
#6	If I'm in work room, then post a tweet. SergioML9 GSI Lab	Rules

Figure 4.4: Ewe Rules Board

Figure 4.5 shows the user page, where the buttons at the top redirect the user to get required data for correct channel function.



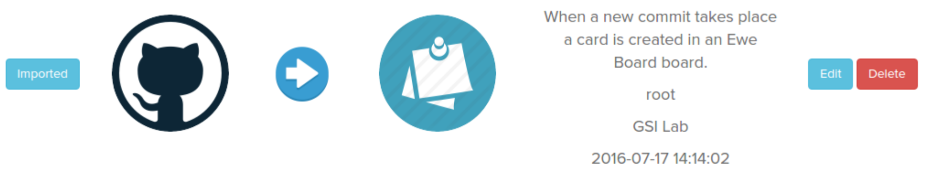
My rules

If I'm in the rest room, then show a toast.
SergioML9
GSI Lab
2016-02-08 17:18:02

If I'm not in the rest room, then switch off TV.

Figure 4.5: User Page Ewe tasker

Next step is importing the rules that automate the tasks the user desires to be automated. This rules are shown in the Rule page which can only be accessed when logged in. The Rule page shows all the rules stored in the Rule Repository and the user must click on the import button in order to take advantage of the rule. An example of how a rule is shown in the Rule page is in the Figure 4.6.



Imported
When a new commit takes place a card is created in an Ewe Board board.
root
GSI Lab
2016-07-17 14:14:02
Edit
Delete

Figure 4.6: Rule Import

Case Study

In this case study, an evaluation of how the system automates several tasks in an agile software development project environment. This test has been done in the GSI[1] laboratory. The following equipment has been used:

- Proximity sensor: 1 estimote beacon placed at the workroom, so it can capture the event of leaving the workroom.
- Mobile Phone: with the Mobile App installed to control the beacon and sending the event to the TAS.
- Internet channels: GitHub channel, Ewe Board channel.
- A computer: where we can access the EAB platform.

These are the conditions assumed for this example:

- The user has done all the previous steps explained in this chapter.
- All channels needed have already been created.
- Rules are already created and imported by the user.
- A beacon is properly configured with the Mobile App, placed in the workroom.

In this context, several tasks will be automated such as:

- Reminding the user of committing changes made when leaving the workroom.
- Posting 's a new card in the repository board if there is a new commit in the GitHub repository.

[1]Intelligent Systems Group, UPM

- Posting a new card if a new issue is created in the GitHub repository and changing the list it is placed on, depending on its status.
- Posting a new card in the repository board if a new member is added to the GitHub repository.

In conclusion, this example shows how the system can automate tasks in an agile software development project environment in order to make it easier for the user to follow the principles of the Scrum methodology. In addition to the task automation, the EAB also provides the user of a platform where information can be shared so that a collaborative and more efficient organization of the project exists. Boards are shared to accomplish this better organization.

CHAPTER 5

Conclusions

This chapter explains the conclusions drawn of the development of the project, and also describes problems encountered , accomplished achievements and lines of future work.

Conclusions

In this project, a platform where all relevant information about a software development project according to the agile methodologies principles has been developed, in order to facilitate the Scrum method for a project. And also the adaptation of a task automation platform has been done, so it is used in an agile software development project environment. The project allows users to automate tasks and also to generate a collaborative organization for a project.

The collaborative organization is achieved through sharing constantly information about the project, this also results in a much more coherent organization. The system provides updated information about the project so that any change in the project is taking into account.

This project is based on a number different technologies and during its development hot to combine and connect all of them has been learnt. The different technologies used are:

- Web technologies:
  - PHP
  - JavaScript
  - HTML5
  - Node.js
- Semantic technologies:
  - Notation3
  - RDF
  - EYE
- Database technologies such as MongoDB.

The software architecture developed in this project is based on two projects: Design and implementation of a Semantic Task Automation Rule Framework for Android Devices [18] and Development of a Task Automation Platform for Beacon enabled Smart Homes [8].

Next sections will describe what goals were achieved by this project, what problems were encountered and also future lines of work.

Achieved goals

These are the goals achieved during the development of this project:

- **Development of an online kanban board application for agile software development projects**. This is the main goal of the project, the development and implementation of a system which facilitates a better agile methodology based organization through kanban boards that show the most relevant information about the project.
- **Adaptation of the task automation platform**. This was a very important goal of the project, the adaptation of the task automation to an agile software development project environment. In order to achieve this several changes have been made to the platform modules, such as the Action Trigger module.
- **Development of a graphical interface that allows users to obtain relevant information about a Scrum project** An interface that shows the information

about an agile software development project and that allows several users to share the information, so that everybody has the same information and a better organization of the project is achieved.

- **Development of a graphical interface that allows users to obtain relevant information about a project GitHub repository**. In this project a graphical interface that shows the user the GitHub repository information for a project is developed. This information is updated automatically.
- **Creation of new EYE rules**. New rules have to be created for the adaptation of the TAS to the new environment. These rules are evaluated by the EYE rule engine, so they have to be written in Notation3, which is a compatible format.
- **New actions triggered**. In order to trigger the actions the EYE response is parsed into JSON format, and after that, this JSON is processed to execute the actions accordingly to the events captured. These actions are executed by the Action Trigger module which has been changed to adapt tothe new actions from the new channels.
- **Connect the EAB with TAS**. The EAB has to connect to the TAS in order to show information related to the TAS in the EAB platform.
- **Connect the TAS with EAB**. The TAS has to connect and store information related to EAB from each user so that the actions of this channel are executed correctly.
- **Connect the TAS with GitHub**. The TAS has to connect and store information related to GitHub from each user so that events coming from GitHub are captured correctly.

Problems faced

The list of the problems encountered during the development of this project is shown bellow:

- EYE : This system uses a rule engine that is still under development, there is not a lot of documentation about it , so some difficulties have been encountered when creating the rules.
- Restyaboard: The application that is used as a base for the EAB platform is also still under development, so changes in the application were made during the development of this project causing some issues.

- Trello: Trello is an online kanban board application which was considered at first for the development of this project but a lot of problems were faced because of it not being an open source application. Its use had to be discarded.
- Connections with Internet channels: This system connects various online applications and during the development of the project problems were faced due to the difficulty to test the system because of the need to have an exposed server.

Future work

Finally, this section lists the various improvements and future lines of work related to this project.

- **Smart Work Place**. This project could be integrated in an smart home environment, so that with sensors a Smart Work Place is developed. This would allow the workers to work more efficiently by having a more comfortable work environment or by the environment reminding the tasks to be done (through smart devices).
- **Adding Scrum Roles to EAB**. An interesting integration for the future is adding Scrum Roles to the EAB platform, so that a project board members can only modify certain lists. This would result in a better solution based on the Scrum principles.
- **Add Internet channels**. In the future it would be useful to add new Internet channels such us Bitbucket or Google Drive so that more information about the project is obtained, resulting in a better more coherent organization.
- **Machine learning integration**. Apply machine learning techniques for suggesting new rules to users. Based on rule popularity or individual parameters.
- **Rule administration in EAB**. In order to make it easier for the user, rule administration could be done frome the EAB platform, so that a user can manage imported rules without changing application.

Bibliography

[1] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries *et al.*, “Manifesto for agile software development,” 2001.

[2] K. Schwaber, *Agile project management with Scrum.* Microsoft press, 2004.

[3] L. Jiang, D.-Y. Liu, and B. Yang, “Smart home research,” in *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*, vol. 2. IEEE, 2004, pp. 659–663.

[4] C. Le Gal, J. Martin, A. Lux, and J. L. Crowley, “Smart office: Design of an intelligent environment,” *IEEE Intelligent Systems*, vol. 16, no. 4, pp. 60–66, 2001.

[5] S. Alliance, “Scrum,” https://www.scrumalliance.org, 2016.

[6] D. Inc., “Docker,” https://www.docker.com/, 2016.

[7] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, “Social coding in github: transparency and collaboration in an open software repository,” in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work.* ACM, 2012, pp. 1277–1286.

[8] S. M. López, “Development of a Task Automation Platform for Beacon enabled Smart Homes,” Master’s thesis, feb 2016.

[9] T. Berners-Lee and D. Connolly, “Notation3 (n3): A readable rdf syntax,” *W3C Submission, Jan*, 2008.

[10] W3C, “Notation3 (n3): A readable rdf syntax,” https://www.w3.org/TeamSubmission/n3/, 2011.

[11] B. De Meester, D. Arndt, P. Bonte, J. Bhatti, W. Dereuddre, R. Verborgh, F. Ongenae, F. De Turck, E. Mannens, and R. Van de Walle, “Event-driven rule-based reasoning using EYE,” in *Joint Proceedings of the 1st Joint International Workshop on Semantic Sensor Networks and Terra Cognita and the 4th International Workshop on Ordering and Reasoning*, Oct. 2015. [Online]. Available: http://ceur-ws.org/Vol-1488/paper-08.pdf

[12] J.DeRoo, “Euler yet another proof engine,” http://eulersharp.sourceforge.net, 2013.

[13] M. Coronado, C. A. Iglesias, and E. Serrano, “Modelling rules for automating the evented web by semantic technologies,” *Expert Systems With Applications*, vol. 42, no. 21, pp. 7979–7990, 2015.

[14] M. Coronado, “Ewe ontology specification,” http://www.gsi.dit.upm.es/ontologies/ewe/, 2016.

[15] MongoDB, "Mongodb," http://searchdatamanagement.techtarget.com/definition/MongoDB, 2016.

[16] Restya, "Restyaboard," http://www.restya.com, 2016.

[17] M. Coronado, C. A. Iglesias, and E. Serrano, "Modelling rules for automating the Evented WEb by semantic technologies," *Expert Systems with Applications*, vol. 42, no. 21, pp. 7979 – 7990, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417415004339

[18] A. Fernández, "Design and implementation of a Semantic Task Automation Rule Framework for Android Devices," Master's thesis, feb 2016.

APPENDIX A

New channels created

This appendix shows the complete new channels created for this project .

- The channel EWE Board is created with the following params:
  - **Title:** EWE Board.
  - **Description:** This channel represents EWE Board.
  - **Nicename:** EWE Board.

Figure A.1: EWE Board Channel

  - **Action:**

  * Title: Post Commit Card.
  * Rule:

```
ewe-board:EWEBoard rdf:type ewe-board:postCommitCard ;
ov:timeCommit ?timeCommit ; # With a time.
ov:authorCommit ?authorCommit ; # With an author.
ov:messageCommit ?messageCommit ; # With a message.
ov:idCommit ?idCommit ; # With a commit ID.
ov:URLCommit ?URLCommit . # With an URL.
ov:RepoCommit ?RepoCommit . # With a repository.
```

  * Prefix:

```
@prefix string: <http://www.w3.org/2000/10/swap/string#>.
@prefix math: <http://www.w3.org/2000/10/swap/math#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    .
@prefix ewe: <http://gsi.dit.upm.es/ontologies/ewe/ns/#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    .
@prefix ov: <http://open.vocab.org/terms/#> .
@prefix ewe-github: <http://gsi.dit.upm.es/ontologies/ewe-
    github/ns/#> .
@prefix ewe-board: <http://gsi.dit.upm.es/ontologies/ewe-
    board/ns/#> .
```

- **Action:**
  * Title: Post Member Card.
  * Rule:

```
ewe-board:EWEBoard rdf:type ewe-board:postMemberCard ;
ov:newmember ?newmember ; # With a name.
ov:repository ?repository . # With a repository.
```

  * Prefix:

```
@prefix string: <http://www.w3.org/2000/10/swap/string
    #>.
@prefix math: <http://www.w3.org/2000/10/swap/math#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns
    #> .
@prefix ewe: <http://gsi.dit.upm.es/ontologies/ewe/ns/#>
    .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns
    #> .
@prefix ov: <http://open.vocab.org/terms/#> .
```

```
@prefix ewe-github: <http://gsi.dit.upm.es/ontologies/
    ewe-github/ns/#> .
@prefix ewe-board: <http://gsi.dit.upm.es/ontologies/ewe
    -board/ns/#> .
```

- **Action:**
  - Title: Post Comment Commit.
  - Rule:

```
ewe-board:EWEBoard rdf:type ewe-board:postCommitCard ;
ov:timeComment ?timeComment ; # With a time.
ov:authorComment ?authorComment ; # With an author.
ov:messageComment ?messageComment ; # With a message.
ov:idCommit ?idCommit ; # With a commit ID.
ov:URLCommit ?URLCommit . # With an URL.
ov:RepoCommit ?RepoCommit . # With a repository.
```

  - Prefix:

```
@prefix string: <http://www.w3.org/2000/10/swap/string
    #>.
@prefix math: <http://www.w3.org/2000/10/swap/math#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns
    #> .
@prefix ewe: <http://gsi.dit.upm.es/ontologies/ewe/ns/#>
     .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns
    #> .
@prefix ov: <http://open.vocab.org/terms/#> .
@prefix ewe-github: <http://gsi.dit.upm.es/ontologies/
    ewe-github/ns/#> .
@prefix ewe-board: <http://gsi.dit.upm.es/ontologies/ewe
    -board/ns/#> .
```

- The channel EWE Board is created with the following params:
  - **Title:** GitHub.
  - **Description:** This channel represents GitHub.
  - **Nicename:** GitHub.
  - **Event:**
    - Title: New Commit.
    - Rule:

Figure A.2: GitHub Channel

```
?event rdf:type ewe-github:newCommit. # Received event of
    type newCommit
?event ewe:timeCommit ?timeCommit. # With a time.
?event ewe:authorCommit ?authorCommit. # With an author.
?event ewe:messageCommit ?messageCommit. # With a message.
?event ewe:idCommit ?idCommit. # With a commit ID.
?event ewe:URLCommit ?URLCommit. # With an URL.
?event ewe:RepoCommit ?RepoCommit. # With a repository.
```

- Prefix:

```
@prefix string: <http://www.w3.org/2000/10/swap/string#>.
@prefix math: <http://www.w3.org/2000/10/swap/math#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
     .
@prefix ewe: <http://gsi.dit.upm.es/ontologies/ewe/ns/#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
     .
@prefix ov: <http://open.vocab.org/terms/#> .
@prefix ewe-github: <http://gsi.dit.upm.es/ontologies/ewe-
    github/ns/#> .
@prefix ewe-board: <http://gsi.dit.upm.es/ontologies/ewe-
    board/ns/#> .
```

- **Event:**
  - Title: New Member.
  - Rule:

```
?event rdf:type ewe-github:newMember. # Received event of
    type newMember
?event ewe:newmember ?newmember. # With a name.
?event ewe:repository ?repository. # With a repository.
```

* Prefix:

```
@prefix string: <http://www.w3.org/2000/10/swap/string#>.
@prefix math: <http://www.w3.org/2000/10/swap/math#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
     .
@prefix ewe: <http://gsi.dit.upm.es/ontologies/ewe/ns/#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
     .
@prefix ov: <http://open.vocab.org/terms/#> .
@prefix ewe-github: <http://gsi.dit.upm.es/ontologies/ewe-
    github/ns/#> .
@prefix ewe-board: <http://gsi.dit.upm.es/ontologies/ewe-
    board/ns/#> .
```

- **Event:**
  * Title: New Comment Commit.
  * Rule:

```
?event rdf:type ewe-github:newCommitComment. # Received
    event of type newCommitComment
?event ewe:timeComment ?timeComment. # With a time.
?event ewe:authorComment ?authorComment. # With an author.
?event ewe:messageComment ?messageComment. # With a
    message.
?event ewe:idCommit ?idCommit. # With a commit ID.
?event ewe:URLCommit ?URLCommit. # With an URL.
?event ewe:RepoCommit ?RepoCommit. # With a repository.
```

  * Prefix:

```
@prefix string: <http://www.w3.org/2000/10/swap/string#>.
@prefix math: <http://www.w3.org/2000/10/swap/math#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
     .
@prefix ewe: <http://gsi.dit.upm.es/ontologies/ewe/ns/#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
     .
@prefix ov: <http://open.vocab.org/terms/#> .
@prefix ewe-github: <http://gsi.dit.upm.es/ontologies/ewe-
    github/ns/#> .
@prefix ewe-board: <http://gsi.dit.upm.es/ontologies/ewe-
    board/ns/#> .
```