Assignment 1

Csc 485A - Topics in Systems

Student Name: Callum Thomas
Student Number: V00828180
Project: Home Assistant
Link to Project Github: https://github.com/home-assistant/home-assistant
Project Website: https://www.home-assistant.io/

Table of Contents

1.0 Introduction

This assignment involves analyzing one of the other group's project's source code by exploring a use case and a growth scenario. In this section, I introduce my chosen project to analyze, and the quality attribute I have chosen to explore in this assignment.

Home Assistant is an open source home automation platform that runs on Python3 [1]. The platform allows you to control devices within your home including Amazon Alexa, Nest, and Google Cast. In addition, the platform allows you to write modules to support whatever devices you choose, extending the platform. The quality attribute I have chosen to examine is Modifiability, with this QA being an architecturally significant requirement of Home Assistant: "The system is built using a modular approach so support for other devices or actions can be implemented easily."[1]. The two quality attribute scenarios I have created in order to analyze Home Assistant are a use-case scenario and a growth scenario. The use-case scenario involves an end-user adding a device to Home Assistant to assess the modifiability of the system. The growth scenario analyzes how the system responds to the users and new devices doubling in number.

2.0 Quality Attribute Scenarios

In this section, I have created template versions of both a use case scenario and a growth scenario for Home Assistant. Within each template, I cover the scenario name, business goals, quality attributes, a stimulus for the scenario, the source of the stimulus, a response from the system, and a response measure.

2.1 Use Case Scenario

Below is a template version of a use-case scenario for the quality attribute of Modifiability. The use case outlines the scenario of a new user wanting to add a new, unsupported device to Home Assistant.

Aspect	Details
Scenario Name	A new user wants to modify Home Assistant in order to support their unsupported device.
Business Goals	To allow users to automate the control of all devices in their home [1]. To support the community to easily extend Home Assistant [1]. To encourage users to edit and add to the documentation of the project [2] and ensure high-quality documentation [2]. To support interoperability of the platform [3].
Quality Attributes	Modifiability.
Stimulus	A new user of Home Assistant requests to modify Home Assistant to support their unsupported home device.
Stimulus Source	A technologically adept homeowner; an end-user of Home Assistant.
Response	The user modifies Home Assistant to support their device, then tests and deploys the modification to Home Assistant's codebase.
Response Measure	The time it takes to make the modification to support a new device should not exceed 8 hours with limited expertise and reading the documentation (to support business goal 2 above). No other components of the platform are broken due to the modification (i.e. no defects are introduced). [13 - Table 7.1]

2.2 Growth Scenario

Below is my template version of a growth scenario for Home Assistant.

Aspect	Details
Scenario Name	Double the amount of end-users and supported devices in one year.
Business Goals	To grow the Home Assistant community [1].

	To support and control all home devices [1].
Quality Attributes	Modifiability.
Stimulus	A sudden influx of new users and new home devices that are yet unsupported.
Stimulus Source	New end-users discovering home assistant and new home devices developed.
Response	More end-users develop support for new devices, and more end-users using Home Assistant in their homes.
Response Measure	Changes in the code do not introduce defects [13 - Table 7.1] and devices are added successfully.

3.0 Analysis

Here, I first analyze the use-case scenario and the growth scenario to assess if they meet the response measures outlined in section 2.1 and 2.2 respectively. To do this, I use diagrams, cited written explanations, and code samples from Home Assistant's source repository. Also recorded are my investigation approaches to both scenarios.

3.1 Use Case Scenario Analysis

In this section, I analyze the use-case scenario outlined in section 2.1, and whether it meets the response measure based on examining the code and the creation of a diagram explaining the use-case in section 3.1.2.

3.1.1 Investigation Approach

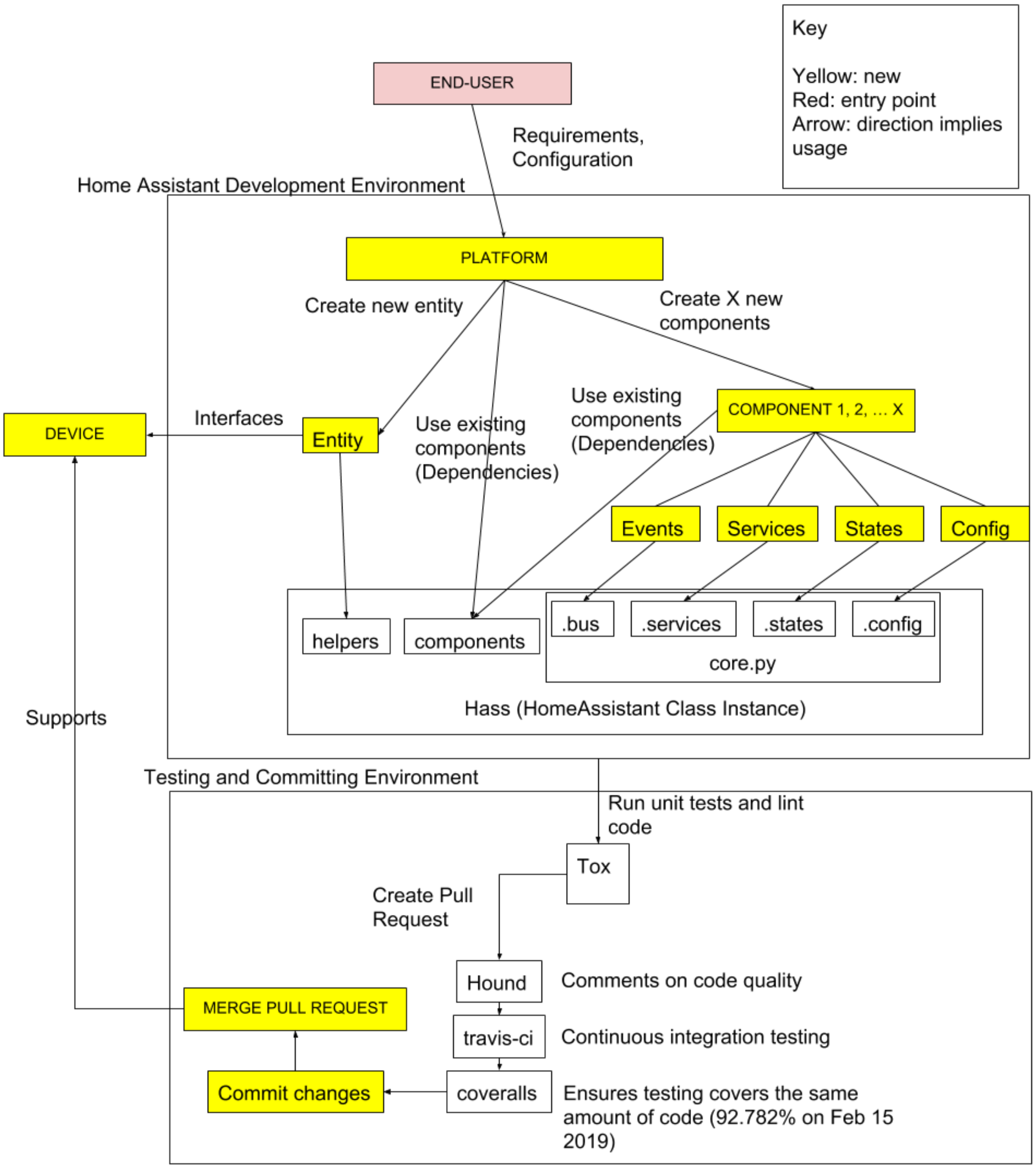
Here I recorded my investigation approach for analyzing the use-case scenario.

1. Read Home Assistant readme.md on Github repository to figure out what Home Assistant is.
2. Visited Home Assistant website: https://www.home-assistant.io/ to determine business goals, and project description.
3. Started reading documentation on the Home Assistant website to get started developing.

4. Cloned the dev branch repository of Home Assistant to my local machine to analyze the code, and ran the system myself in a development environment context, like the end-user trying to modify the system in the use-case scenario would do.
5. Followed documentation instructions to create a component and a platform.
6. Examined existing components such as the AppleTV and Nest to judge the modifiability of the code, by determining dependencies, coupling, and cohesion.
7. Read the documentation on Home Assistant test plans to assess the requirements for submitting code to Home Assistant.
8. Summarized my investigation in order to answer if the scenario in section 2.1 meets the response measure.

3.1.2 Use-Case Scenario Diagram

Below is a diagram representing the use case scenario in section 2.1. The red box “end-user” is the entry point for the diagram, so you should start reading it there, and follow the arrows. The arrows indicate that a structure is using another structure for the purpose that is annotated. Yellow boxes indicate new elements to the system. In this case, the user is creating a new platform, which introduces new components. The device is unsupported, so it is also new to the system. The diagram is split into two environments: the Home Assistant development environment which encapsulates all of the Home Assistant systems, and the testing/committing section which includes the processes needed to test and then commit code changes into the repository.


Key
Yellow: new
Red: entry point
Arrow: direction implies usage
END-USER
Requirements, Configuration
Home Assistant Development Environment
PLATFORM
Create new entity
Create X new components
DEVICE
Interfaces
Entity
Use existing components (Dependencies)
Use existing components (Dependencies)
COMPONENT 1, 2, ... X
Events
Services
States
Config
helpers
components
.bus
.services
.states
.config
core.py
Hass (HomeAssistant Class Instance)
Supports
Testing and Committing Environment
Run unit tests and lint code
Tox
Create Pull Request
Hound
Comments on code quality
travis-ci
Continuous integration testing
MERGE PULL REQUEST
Commit changes
coveralls
Ensures testing covers the same amount of code (92.782% on Feb 15 2019)

3.1.3 Analysis

This section outlines my reasoning of the use case scenario in section 2.1 by using the diagram in section 3.1.2 and by analyzing Home Assistant's codebase and documentation.

3.1.3.1 Analysis of Modifiability

In this section, I analyze the use case scenario defined in section 2.1 using the diagram in 3.1.2, Home Assistant documentation, and source code.

As an architecture consultant, I conclude that the existing code meets the response measure of the modifiability use-case quality attribute scenario outlined in section 2.1. By reading documentation [4] describing how to get started developing Home Assistant, I have mapped the classes that are available to the end-user to construct support for a new device. The main concept of device support in Home Assistant is the Component, coloured yellow in the 3.1.2 diagram. A component encapsulates the Home Assistant logic to control a device such as a Nest sensor monitoring temperature in your home. The logic is made up of events (data emitted by the event bus [7]), services (controlling component behavior [9]), states (entity (device) state [8]), and configuration (component configuration [10]), these aspects are shown on the diagram (section 3.1.2) mapping between the component and the Home Assistant system “Hass”.

The component can only control devices by using an Entity, which is an abstract class with unimplemented methods for controlling the real-world device, and the Entity must do so using third-party python libraries. The reasoning for this is explained in the documentation: “One Home Assistant rule is that platform logic should never interface directly with devices. Instead, use a third-party Python 3 library. This way, Home Assistant can share code with the Python community and keep the project maintainable.”[6]. This design choice also supports modifiability by reducing the time it takes to add new features because the code to interface devices is already written. The modifiability cost of introducing defects depends on the third party library’s code, which may cause problems if the third party libraries introduce defects on their end, however, the defect will only affect entities that use that library, not other entities. For the 2.1 use case scenario, an entity must be written to control the new device. This should be fast to implement as the abstract class is already written, meeting the response measure of not taking excessive time to add new features.

In total, all of these elements are then encapsulated by the Platform pattern. Platforms use Components, Entities, and other Platforms to set up user-made logic and control devices using configurations, states, events, and services. The platform is shown in the diagram at the top as it encapsulates all other modules below.

To add support for a new device, the process is outlined in the diagram [Section 3.1.2]. First, the user creates a new platform. To do this, they must specify a configuration, which includes technical configurations such as hostnames for networking. Then, they specify requirements, which are the python libraries that the platform uses to function. Then, the user can create the platform using existing components, which are classes that support different devices such as media players, lights, switches. Each component uses Hass, which is an instance of the Home Assistant class allowing access to the entire Home Assistant system: to listen to events (eg. a change in temperature detected), use services (controlling components, eg. turning them on and off), states (the current state of a component eg. current temperature reading), and config (user made configurations for components, eg. max temperature allowed). Components may be switched out and removed easily, as each component is a module, encapsulating all of its functions and keeping its own dependencies. For example, if your new device used a Nest sensor, you can easily add it to your new component by using the lines:

```
from homeassistant.components import nest
DEPENDENCIES = ['nest']
```

The Nest component is already created by other users of the open source project. Home Assistant also has pre-made generic low-level components such as the light, switch, and media_player that are imported in the same fashion in higher level components such as AppleTV. You can easily modify Home Assistant by mixing and matching these components and then specifying your own behaviors by using the Hass object (event bus, states, config, and services). This supports the response measure of the use case scenario because it is fast to create new components by reusing code, as most new components will use already existing components. If you want to create an entirely new component, the Hass object is available to the end-user to define events, services, configuration, and states; this is shown in section 3.1.2 under the new component structure. Component creation is also sped up by the helpers collection, which is a set of helper functions that implement frequently used functions of Home Assistant, such as the discovery function which discovers available platforms in your home.

3.1.3.2 Analysis of Testing and Committing Changes

Here I analyze the testing and committing pipeline of Home Assistant, and whether it supported the response measure.

Finally, once the user has created a new platform/component following the guidelines in the documentation, the code can be committed to Home Assistant's repository as a new component. First, however, the code must pass both unit tests and checks from linting tools, to ensure changes do not break any part of the system. To do this, Tox, which is an automated testing framework for Python [11], is included in Home Assistant's development branch to automatically run unit tests prior to committing code. Running "Tox" in the root project directory will automatically run all tests found in the tests folder, ensuring that changes do not break existing functionality. This feature of the architecture satisfies the response measure in section 2.1 stating that changes should not break other components. Next, to submit code to Home Assistant a pull request should be made. Home Assistant's git repository has implemented some automated checks before pull requests can be merged, these are shown in the Appendix [1]. On top of Tox, these tools such as Travis-CI help ensure that code changes do not introduce defects by further testing code before merges.

3.1.4 Conclusion

Below is my conclusion to the analysis of the use-case scenario.

This architecture supports modifiability because little code is needed to get started making a component, fulfilling the response measure that required non-excessive time to make changes in the system. There are also helper functions to speed up development, providing frequently used function support. Documentation was clear and easy to follow, I created a “hello world” component within an hour, after setting up my development environment. Next, to actually contribute this code to the project, Home Assistant uses “Tox” to facilitate acceptance tests. This ensures that modifications do not break the code by running unit tests and linting the code, ensuring that the response measure is met.

3.2 Growth Scenario Analysis

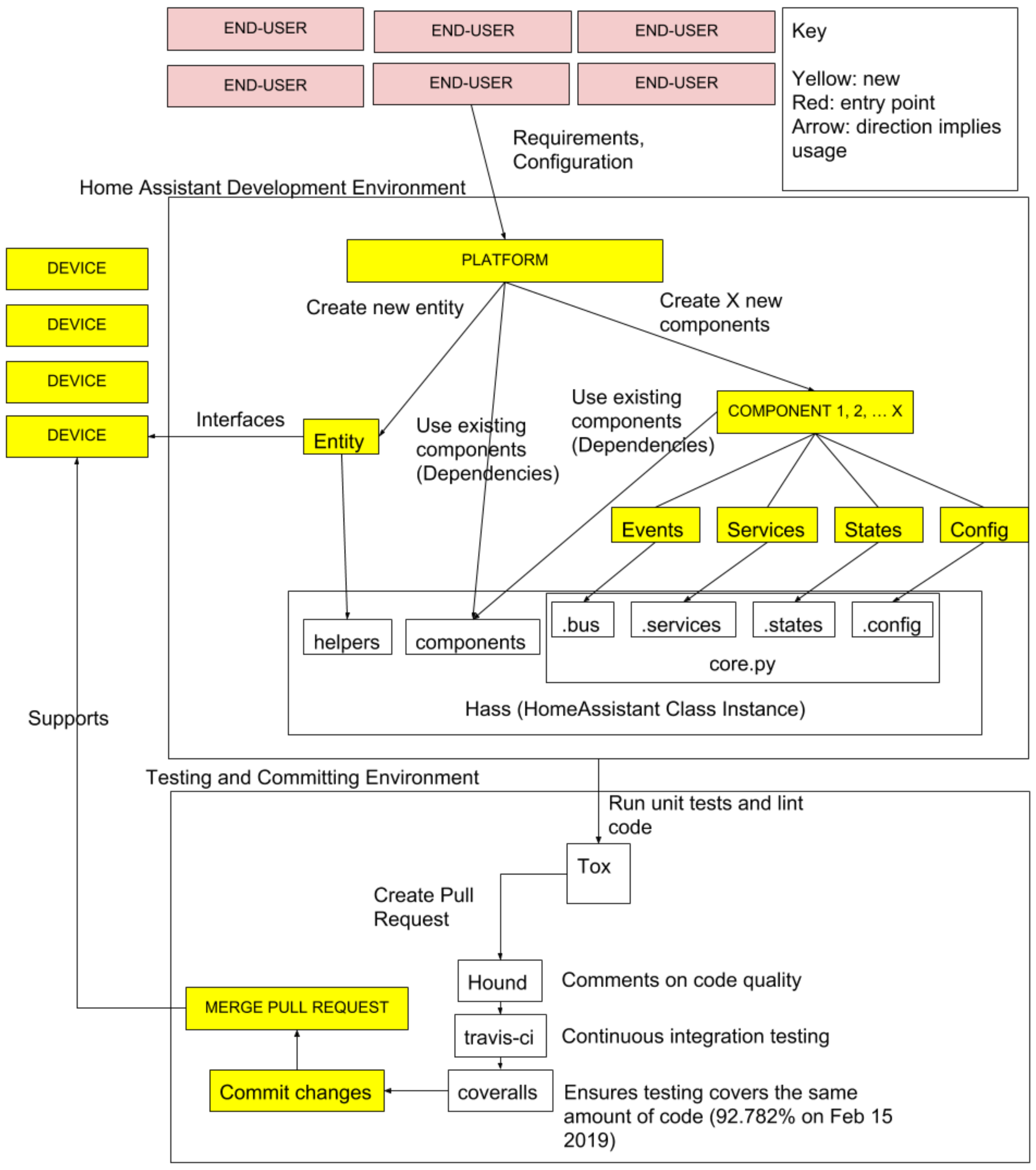
Here I analyze the growth scenario, in which the number of users and devices to be supported is doubled.

3.2.1 Investigation Approach

1. Investigate deployment pipeline of Home Assistant.
2. Research continuous integration used by Home Assistant.
3. Analyze test plans of Home Assistant.

3.2.2 Growth Scenario Diagram

Below is my diagram depicting the growth scenario in which the number of end-users is doubled as well as the number of new devices to be added to Home Assistant.



3.2.3 Analysis

Here, I analyze the growth scenario and determine if it fulfills the response measure outlined in section 2.2.

The growth scenario (section 2.2) states that the number of users and devices to add to Home Assistant have doubled. With user growth the following challenges are presented to this Home Assistant architecture:

1. Increased number of contributors to the open source project.
2. Increased number of pull requests.
3. Increased number of components.

The diagram in section 3.2.2 shows the increased number of users, and then the increased number of new devices being added to Home Assistant. The diagram is identical to the use case scenario diagram in 3.1.2 otherwise. The diagram implies that each user has their own "copy" of the full diagram, showing their own pull requests, tox testing, and development. As a result, there is an increased number of contributors, pull requests, and components in the system.

One of the business goals of the project is to grow the community (section 2.2 - business goals), so this growth would be seen as a positive thing for Home Assistant. However, the increased number of contributors could be problematic in terms of the open source project. More contributors can lower the overall code quality and introduce bugs to the system because there is more code being written and more chances of things going wrong. Luckily, as I have analyzed in the use-case scenario, Home Assistant has a number of precautions to prevent this from happening. Tox, the automated Python testing framework, is enforced for commits, meaning that unit tests and linting must pass before the code is contributed to the project. This alone should prevent bad code from getting into the project as a result of the growth scenario. In addition, tools such as Hound, Travis-CI, and Coveralls are integrated into the pull request pipeline of Home Assistant's repository. These tools support continuous integration that we would see with the increased number of users, fulfilling the response measure of the growth scenario to prevent defects from being introduced.

The increase in new devices should work as expected due to the existing architecture of the project. Described earlier, the only way to interface devices is through the Entity class, and third-party library usage is enforced. This allows reusing library code to support new components, meaning that it is more likely that new devices will be added successfully. The existing components should help the end-users develop new components, as the old components are all built on top of low-level components such as the light, switch, and media_player.

3.2.4 Conclusion

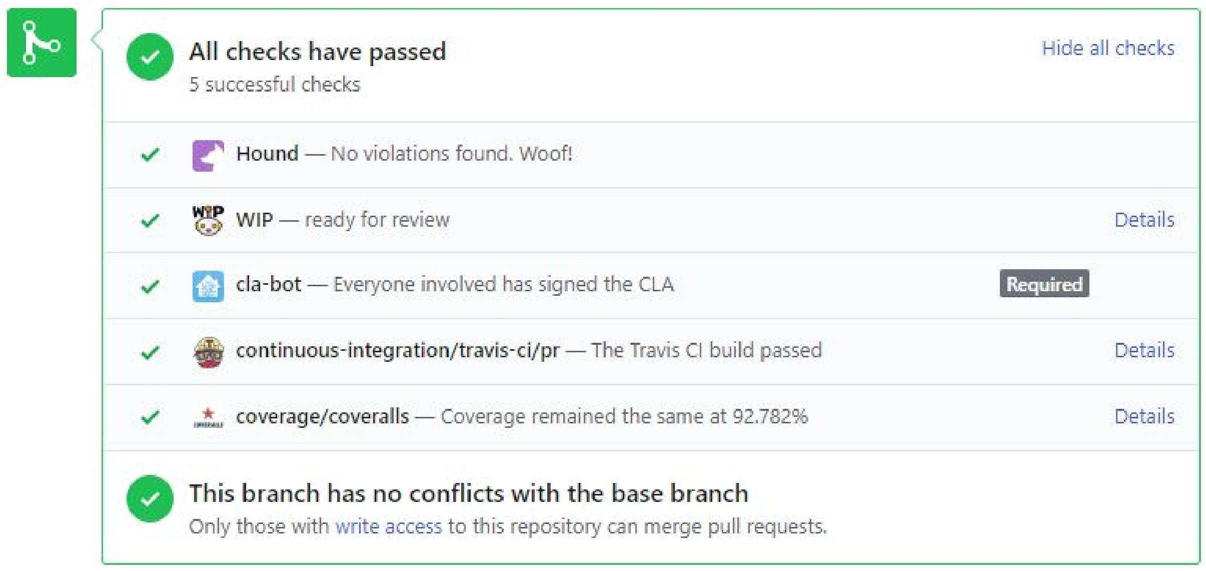
Following the use-case scenario, I analyzed the modifiability of the architecture, including the processes outlined for committing and testing the changes. Given that current procedures are followed, I conclude that doubling the users and new devices to add to Home Assistant will meet the response measure outlined in section 2.2: changes in the code do not introduce defects and devices are added successfully. As long as testing procedures and continuous integration guidelines are followed, Home Assistant should have no problem growing the number of users and devices exponentially.

Sources

[1] Github page for Home Assistant https://github.com/home-assistant/home-assistant
[2] Home Assistant - State of the Union 2018 - Amsterdam, Nov 2018 @ ING
https://www.youtube.com/watch?v=egcOCWIh9jQ
[3] Google slides presentation from the State of the Union 2018 presentation:
https://docs.google.com/presentation/d/1ZfmDOhBZJVcsgc-Yr2GVoenia5LYf9v2wYTy92zqJJY/edit#slide=id.p
[4] Home Assistant development guide:
https://developers.home-assistant.io/docs/en/dev_101_index.html
[5] Home Assistant architecture:
https://developers.home-assistant.io/docs/en/architecture_index.html
[6] Home Assistant documentation for creating a new platform:
https://developers.home-assistant.io/docs/en/creating_platform_index.html
[7] Home Assistant events:
https://developers.home-assistant.io/docs/en/dev_101_events.html
[8] Home Assistant states:
https://developers.home-assistant.io/docs/en/dev_101_states.html
[9] Home Assistant services:
https://developers.home-assistant.io/docs/en/dev_101_services.html
[10] Home Assistant config:
https://developers.home-assistant.io/docs/en/dev_101_config.html
[11] Tox Python Testing Framework:
https://tox.readthedocs.io/en/latest/
[12] Home Assistant style guidelines:
https://developers.home-assistant.io/docs/en/development_guidelines.html
[13] Software Architecture in Practice (3rd Edition) (SEI Series in Software Engineering) 3rd edition. Paul Clements, Rick Kazman

Appendix

[1] Github pull request checks https://github.com/home-assistant/home-assistant/pull/21053



All checks have passed
5 successful checks

✓ Hound — No violations found. Woof!

✓ WIP — ready for review

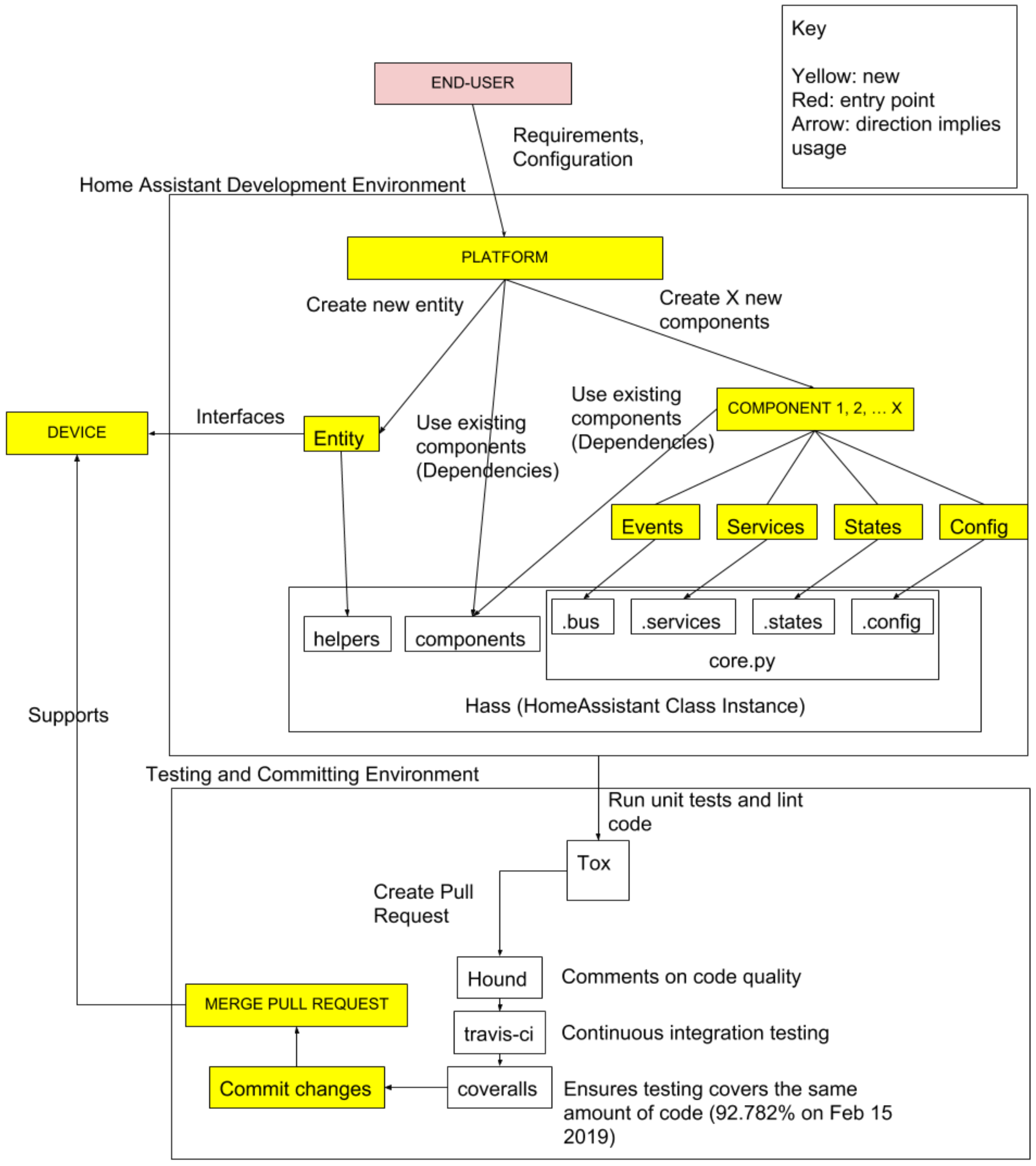
✓ cla-bot — Everyone involved has signed the CLA

✓ continuous-integration/travis-ci/pr — The Travis CI build passed

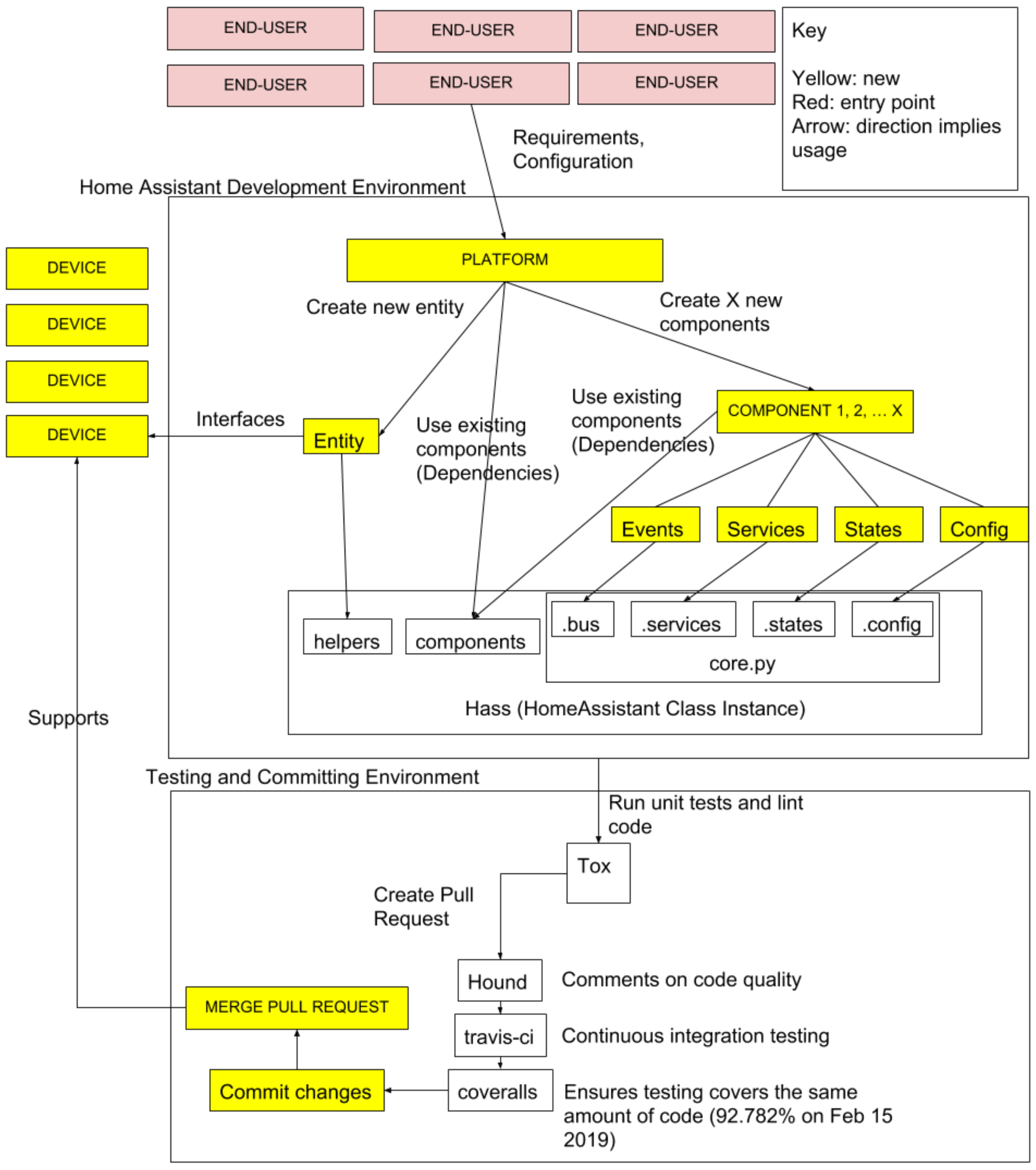
✓ coverage/coveralls — Coverage remained the same at 92.782%

This branch has no conflicts with the base branch
Only those with write access to this repository can merge pull requests.

[2] Use-case scenario diagram


Key
Yellow: new
Red: entry point
Arrow: direction implies usage
END-USER
Requirements, Configuration
Home Assistant Development Environment
PLATFORM
Create new entity
Create X new components
Use existing components (Dependencies)
COMPONENT 1, 2, ... X
DEVICE
Interfaces
Entity
Use existing components (Dependencies)
Events
Services
States
Config
helpers
components
.bus
.services
.states
.config
core.py
Hass (HomeAssistant Class Instance)
Supports
Testing and Committing Environment
Run unit tests and lint code
Tox
Create Pull Request
Hound
Comments on code quality
MERGE PULL REQUEST
travis-ci
Continuous integration testing
Commit changes
coveralls
Ensures testing covers the same amount of code (92.782% on Feb 15 2019)

[3] Growth scenario diagram


END-USER
END-USER
END-USER
END-USER
END-USER
END-USER
Key
Yellow: new
Red: entry point
Arrow: direction implies usage
Requirements, Configuration
Home Assistant Development Environment
PLATFORM
Create new entity
Create X new components
DEVICE
DEVICE
DEVICE
DEVICE
Interfaces
Entity
Use existing components (Dependencies)
Use existing components (Dependencies)
COMPONENT 1, 2, … X
Events
Services
States
Config
helpers
components
.bus
.services
.states
.config
core.py
Hass (HomeAssistant Class Instance)
Supports
Testing and Committing Environment
Run unit tests and lint code
Tox
Create Pull Request
Hound
Comments on code quality
travis-ci
Continuous integration testing
coveralls
Ensures testing covers the same amount of code (92.782% on Feb 15 2019)
MERGE PULL REQUEST
Commit changes