UNIVERSITEIT ANTWERPEN

Academiejaar 2015-2016

Faculteit Toegepaste Ingenieurswetenschappen

II-Distributed Systems

SmartCity Project

Huybrechts Thomas, Janssens Arthur, Joosens Dennis, Vervliet Niels

Prof. Hellinckx Peter, Paillet Eric, Vercauteren Charles, De Bock Yorick

Master of Science in de industriële wetenschappen: Electronica-ICT

Universiteit Antwerpen

SmartCity Project – V0.3

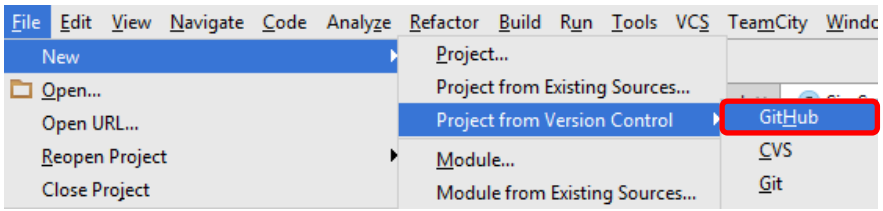
Inhoudsopgave

1 Installatie instructies

1.1 IntelliJ

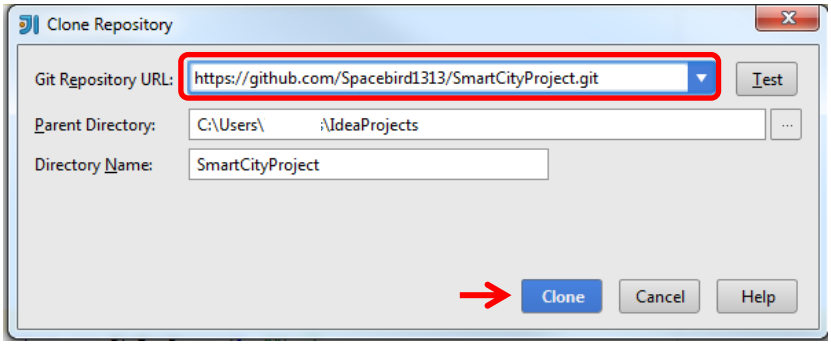
Voeg de repository ‘SmartCity Suite’ als project toe aan de IntelliJ IDE. Maak hiervoor een nieuw project aan volgens de onderstaande instructies.

> File > New > Project from Version Control > GitHub



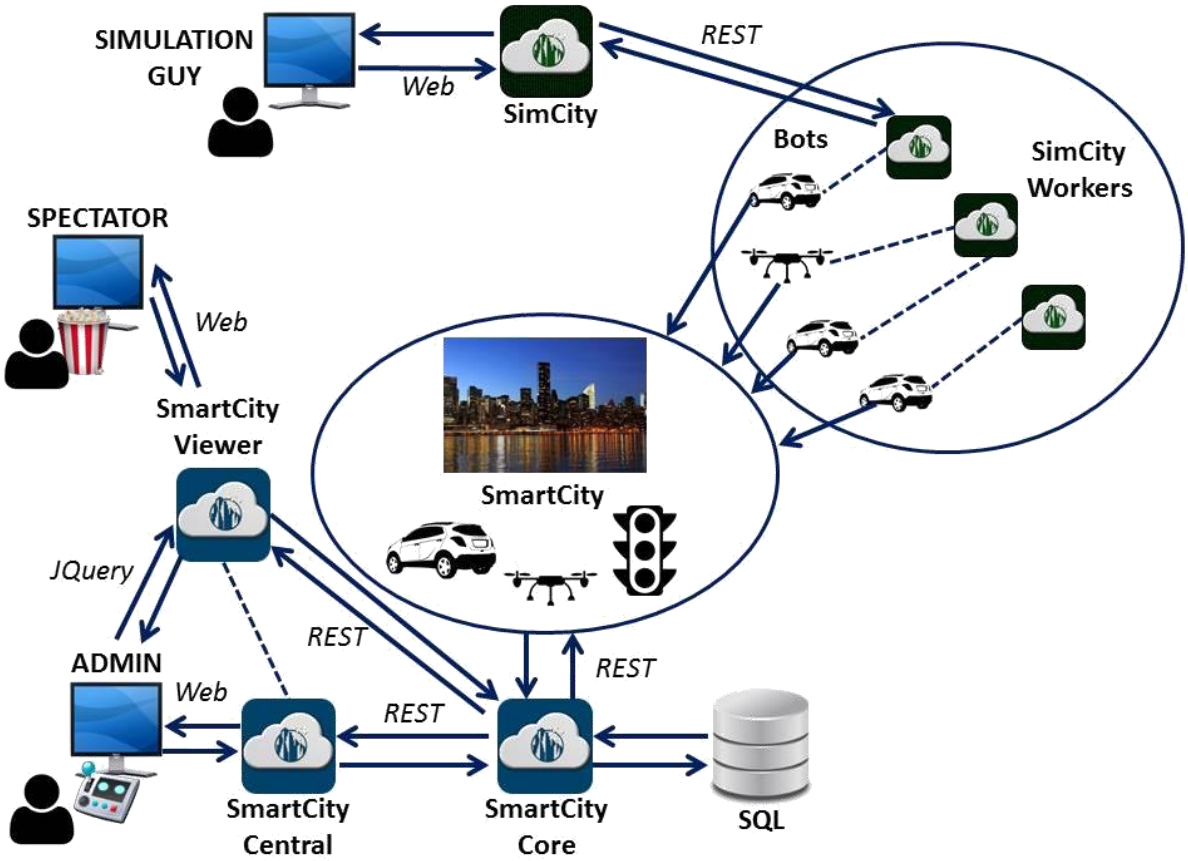
Voeg in het nieuwe venster de volgende gegevens van de server in:

URL: https://github.com/Spacebird1313/SmartCityProject.git



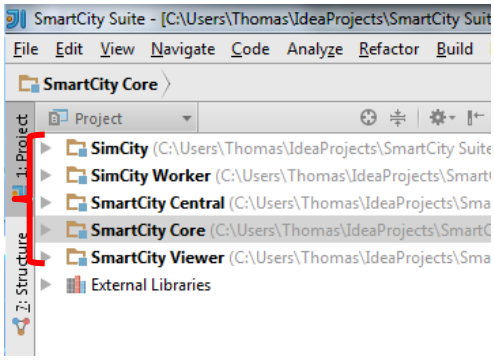
Klik op de knop ‘Clone’ om het project te importeren. Het project is nu beschikbaar in IntelliJ.

1.2 Project structuur



Projectmodules:

- SmartCity Core: Core control system of the SmartCity project.
- SmartCity Central: Visual front-end to control the entire SmartCity project.
- SmartCity Viewer: Visual front-end simulator of Smart City.
- SimCity: Simulator front-end to control SimCity Workers.
- SimCity Workers: Simulator worker to control simulator threads.

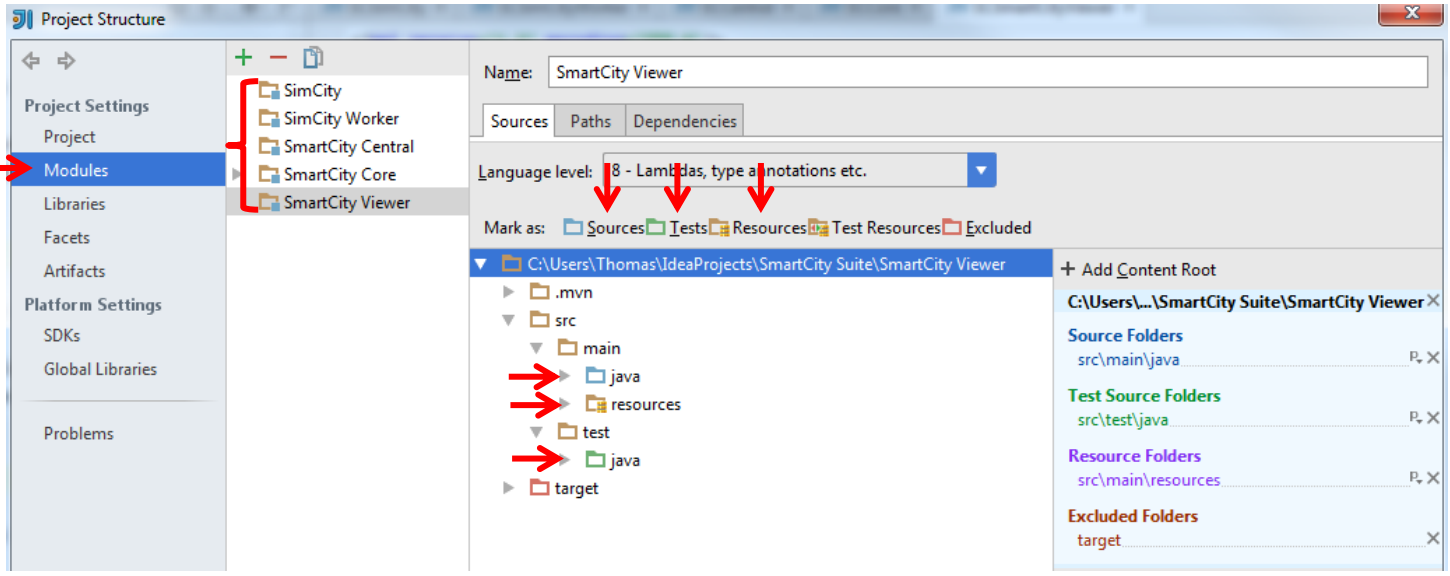


Indien de bovenstaande structuur incompleet is, dan kan dit ingesteld worden via ‘Project Structure…’ menu. *>File >Project Structure…*

Per module dienen er drie source mappen aangeduid te worden:

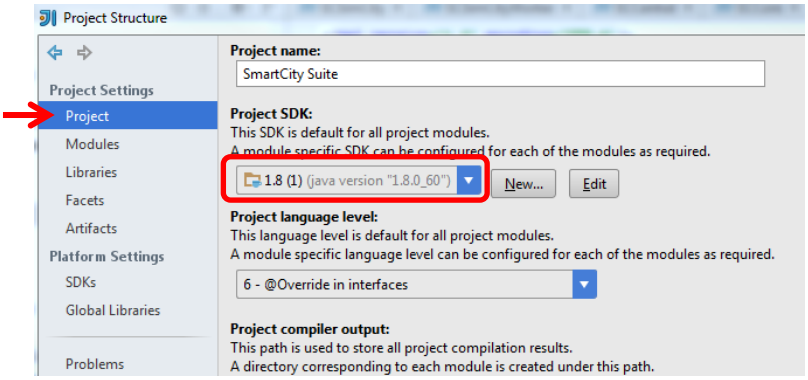
- Sources
- Tests
- Resources

Deze mappen geven een aanduiding aan IntelliJ/Maven waar de diverse soorten source bestanden aanwezig zijn in het project.

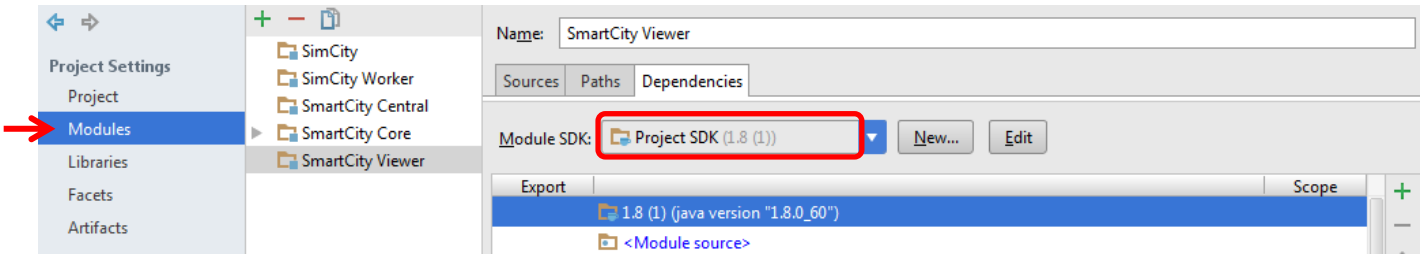


Duidt de bovenstaande mappen aan als root folder voor elke type van resource. De onderliggende mappen volgen de structuur van de ingestelde group id (vb: be.uantwerpen.sc → java\be\uantwerpen\sc).

Indien de compiler SDK niet ingesteld is voor één of meerdere modules, dan kan dit geselecteerd worden in het menu 'Project'.



Indien nodig, dient de SDK nog aangeduid te worden in de project modules. Kies hiervoor terug het menu 'Modules' en de tab bovenaan 'Dependencies'. Duid in het veld 'Module SDK' de optie 'Project SDK' aan.

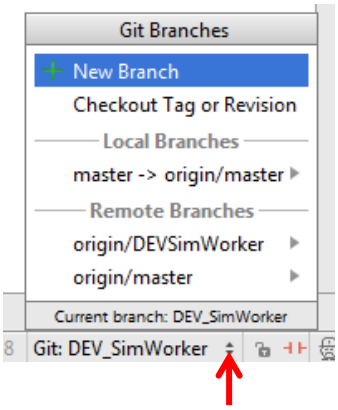


In dit project wordt er gebruik gemaakt van ***Oracle Java SDK versie 1.8.60 of later***.

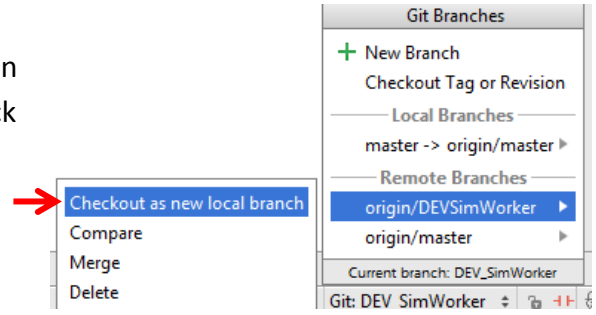
1.3 Branches

Rechts onderaan in de statusbalk van IntelliJ wordt de huidige branch aangeduid waarin de developer zit te werken en waarna hij/zij zijn code zal committen.

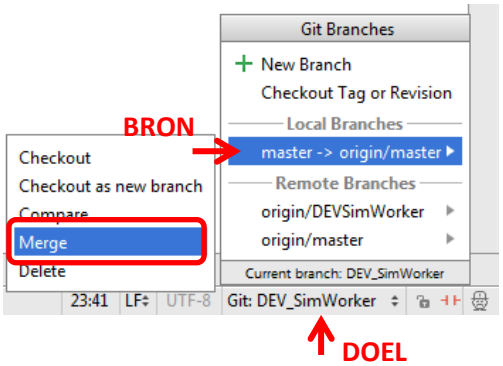
Bij het openklikken van dit menu heeft men de opties om een nieuwe branch af te takken van een andere branch, te wisselen tussen verschillende branches of twee branches te vergelijken/te mergen.



Om een remote branch (server-versie) naar uzelf te trekken om deze te kunnen bewerken, selecteert u de optie ‘Check out as local branch’.



Let op! Bij het mergen tussen twee branches zullen de verschillen tussen de twee branches steeds van de geselecteerde branch naar de huidige geopende branch worden overgenomen.



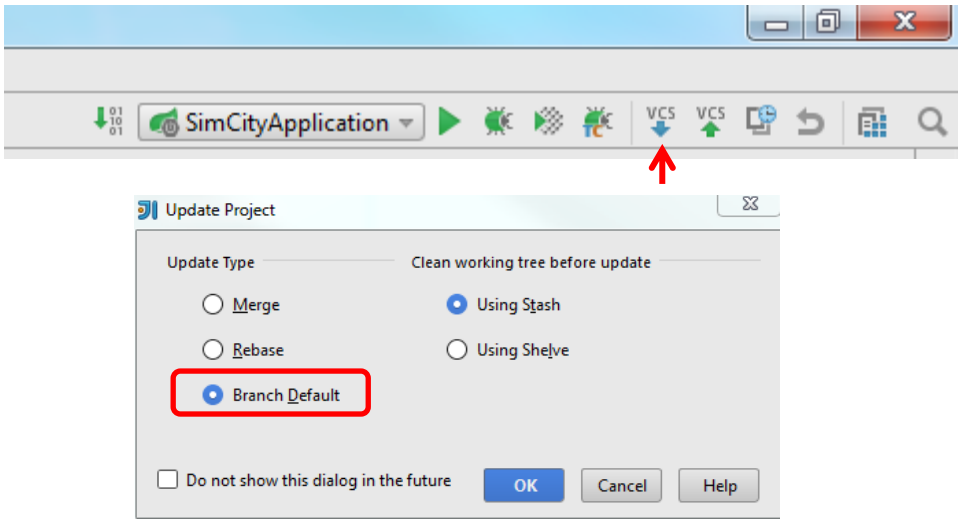
Indien de 'Remote Branch' lijst niet meer up-to-date is, kan deze vernieuwd worden met een 'fetch' commando. Deze optie staat onder: *>VCS > Git >Fetch*

De eerste stap om wijzigen toe te voegen aan Git, is deze aanpassingen lokaal te committen.

Vul bij elke commit een duidelijke gestructureerde commentaar bij over wat er concreet in de code is gewijzigd. Maak er een goede gewoonte van om vaak kleine commits te maken. Hierdoor is het veel eenvoudiger om bij fouten een paar commits terug te keren zonder dat er veel moeite verloren gaat en kunnen problemen sneller gevonden worden.

Om de lokale wijzigingen op de server te delen voor anderen, dienen de commits nog 'gepushed' te worden naar de server. Dit kan uitgevoerd worden via het menu: *> VCS > Git > Push…*

De omgekeerde richting is ook mogelijk. Gebruik hiervoor de optie 'pull' of 'update', let hierbij op dat u de default settings voor 'Update Type' niet wijzigt! (Branch Default)



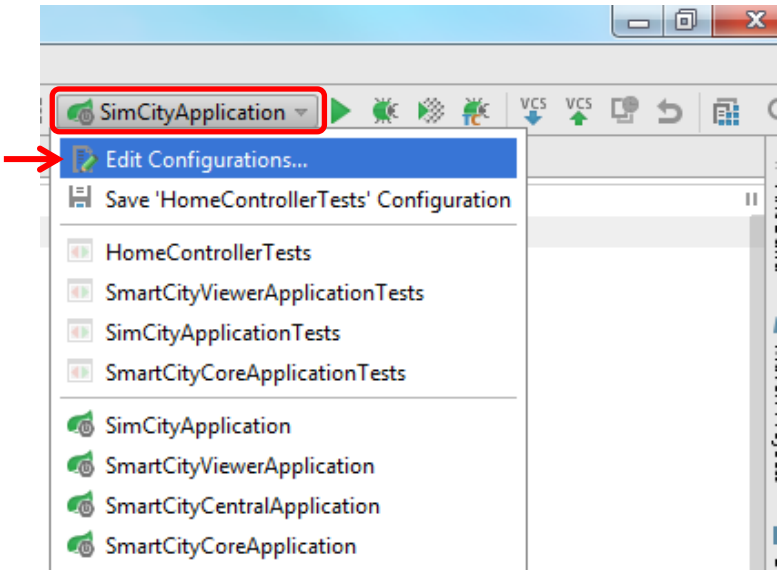
Let op! Bij het uitvoeren van een commit mogen **geen nieuwe files** (met groen aangeduid in de commit lijst) uit de **'.idea' map** worden toegevoegd aan de repository! Alle noodzakelijke files om het project te reconstrueren (met blauw aangeduid in de commit lijst) zijn reeds aanwezig op de Git. Elke andere file is strikt persoonlijk voor uw configuratie en zal bij de anderen IDE's leiden tot conflicten!

1.4 Uitvoeren van een module

Elke module is ontworpen als een Spring Boot Servlet. Om het ontwerp en deploy proces te vereenvoudigen, is er in de modules een configuratie ingebouwd die het mogelijk maakt te wisselen van context door middel van een VM argument.

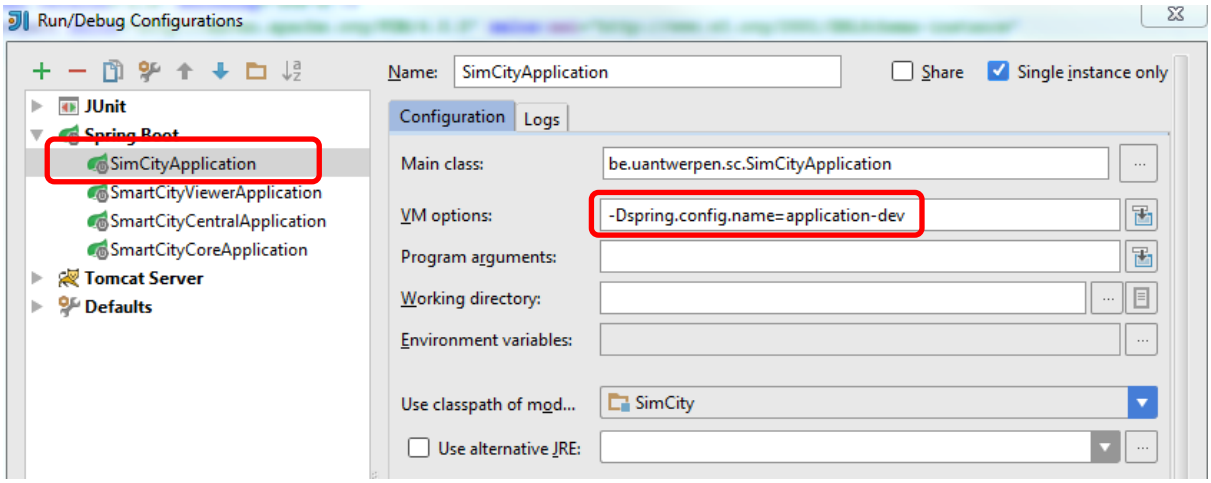
De default configuratie van elke module (zonder optie) zal steeds een servlet applicatie starten die kan uitgevoerd worden op een Java container server (vb Tomcat). Daarnaast zullen modules met een database connectie trachten verbinding te maken met de externe database. Voor deze modus moeten geen opties toegevoegd worden. Deze configuraties zijn te vinden in het bestand 'application.properties' in de resource folder.

In de development configuratie zal bij het uitvoeren van de applicatie een embedded Tomcat worden geïnstantieerd waarin de servlet snel kan gedeployed en getest worden. Daarnaast zullen modules met een database connectie een eigen in-memory database starten die lokaal geraadpleegd kan worden tijdens het debuggen. Om de debug configuratie in te stellen, opent u de run configuraties.



Vul in het 'VM options' veld het onderstaande argument in:

-Dspring.config.name=application-dev



Met deze optie wordt de development modus van de module geactiveerd. Om de instellingen van deze modus te bekijken of te wijzigen, dient u de 'application-dev.properties' file te bekijken in de resource folder.

Een derde standaard configuratie is de standalone mode. In deze modus zullen de modules uitgevoerd worden voor de productie omgeving zoals de default configuratie. Enkel zal in deze modus de module gestart worden binnen een embedded Tomcat server gelijkend aan development mode. Deze mode kan gestart worden met de VM argument:

-Dspring.config.name=application-standalone

De configuratie van deze modus kan bekeken en gewijzigd worden in de overeenkomstige configuratiebestand: 'application-standalone.properties'.

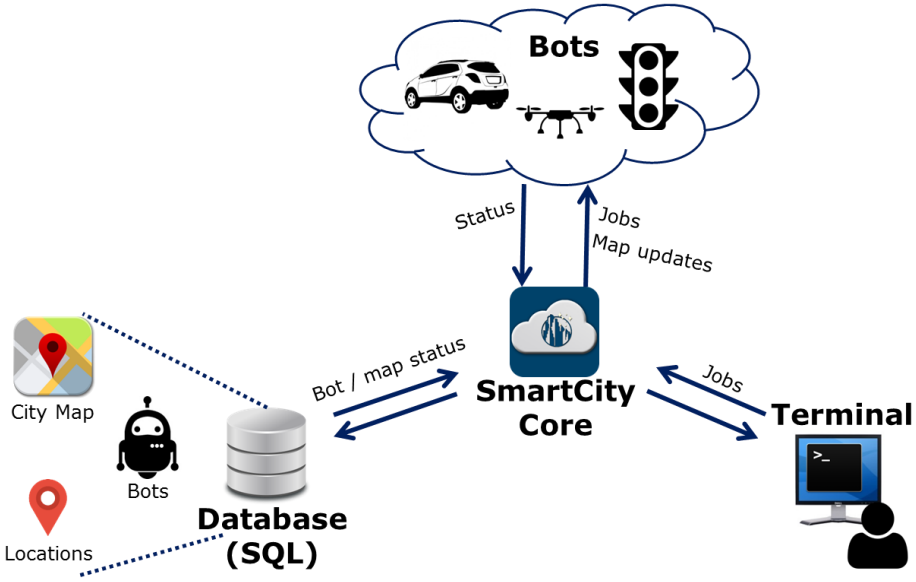
Let op! Om een VM argument mee te geven bij het uitvoeren van een module in terminal, dienen deze argumenten voor de optie '-jar' geplaatst worden. Hieronder wordt een voorbeeld gegeven om de SmartCity Core uit te voeren in development mode in een terminal.

 # java -Dspring.config.name=application-dev -jar SCCore-x.x.x-RELEASE.jar

2 SmartCity Modules

2.1 SmartCity Core

De SmartCity Core is het 'hart' van het gedistribueerde systeem. De module beheerd de toegang tot de database. Andere modules kunnen status updates pushen naar of opvragen van de server. Voorbeelden van modules zijn: SmartCity Viewer en SmartBots. Om load balancing toe te laten, is dit systeem stateless ontworpen.

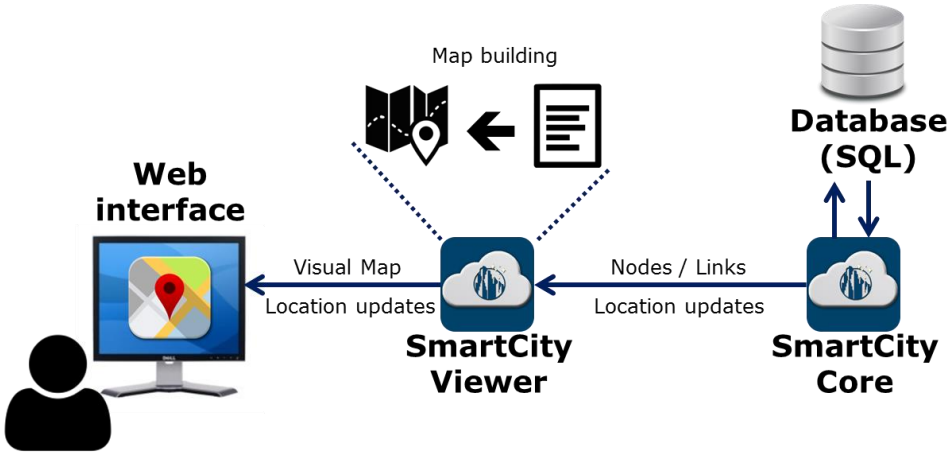


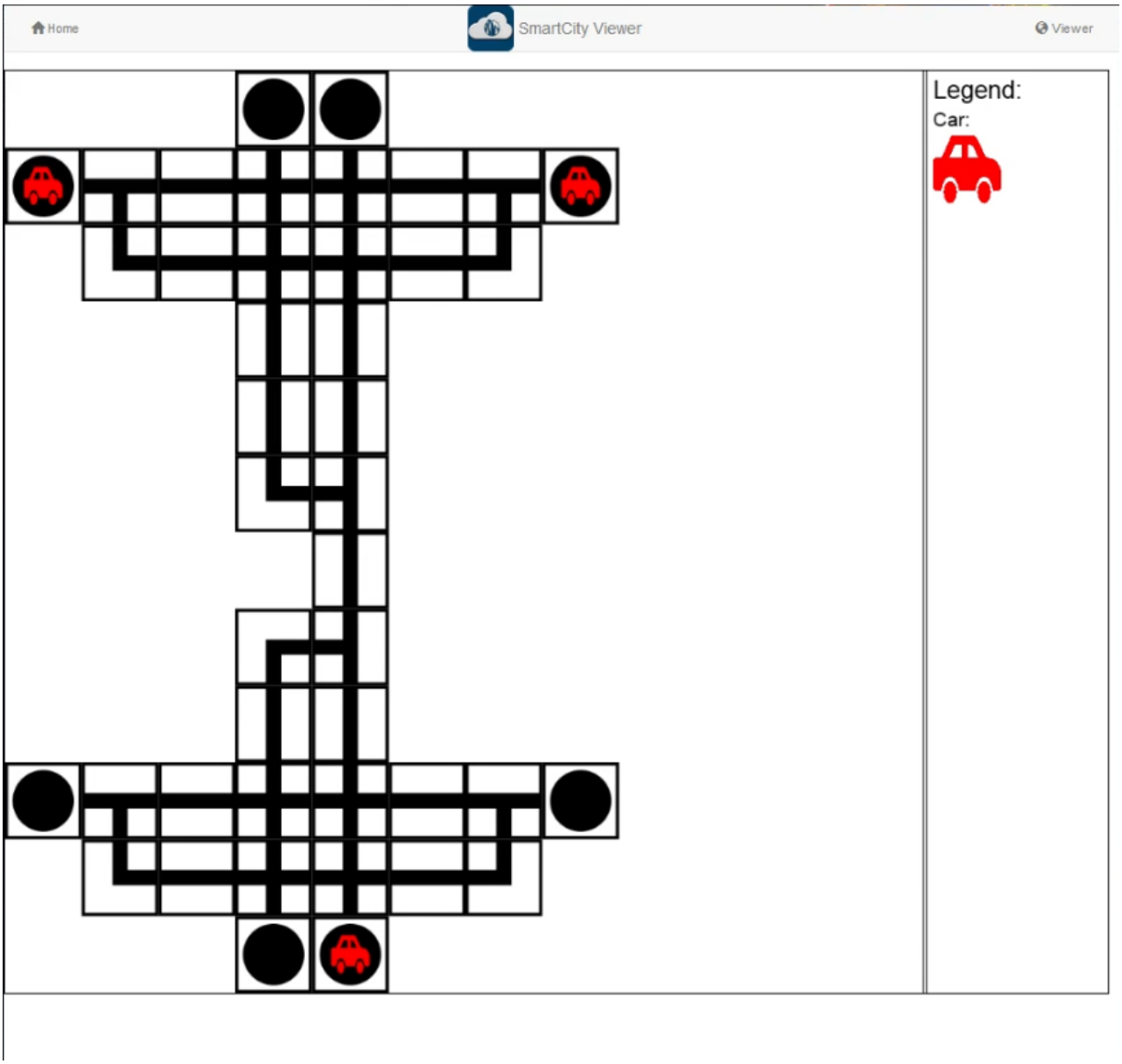
2.2 SmartCity Central

De SmartCity Central module is een webservice die een web front-end aanbied om de status van de SmartCity Core op te vragen in een overzichtelijke gebruiksvriendelijke dashboard. Daarnaast zal deze de mogelijkheid aanbieden om jobs te pushen naar bots, zoals een navigatie job naar een voertuig bot. In de huidige implementatie van dit project is de Central module nog niet functioneel geïmplementeerd.

2.3 SmartCity Viewer

De SmartCity Viewer creëert een visuele voorstelling van de fysieke map. Hiervoor zal de viewer applicatie de map opvragen op de SmartCity Core. Uit de informatie van de verschillende linken en hun lengte wordt een mogelijke voorstelling van de reële map opgebouwd. Op de aangeboden web front-end worden alle bots aanwezig in de SmartCity aangeduid. De web front-end is compatibel met elk toestel met een compatibele webbrowser. De web lay-out wordt aangepast aan het schermresolutie van het toestel, zoals pc, tablet of smartphone.


Map building
Database
(SQL)
Web
interface
Visual Map
Location updates
Nodes / Links
Location updates
SmartCity
Viewer
SmartCity
Core


Home
SmartCity Viewer
Viewer
Legend:
Car:

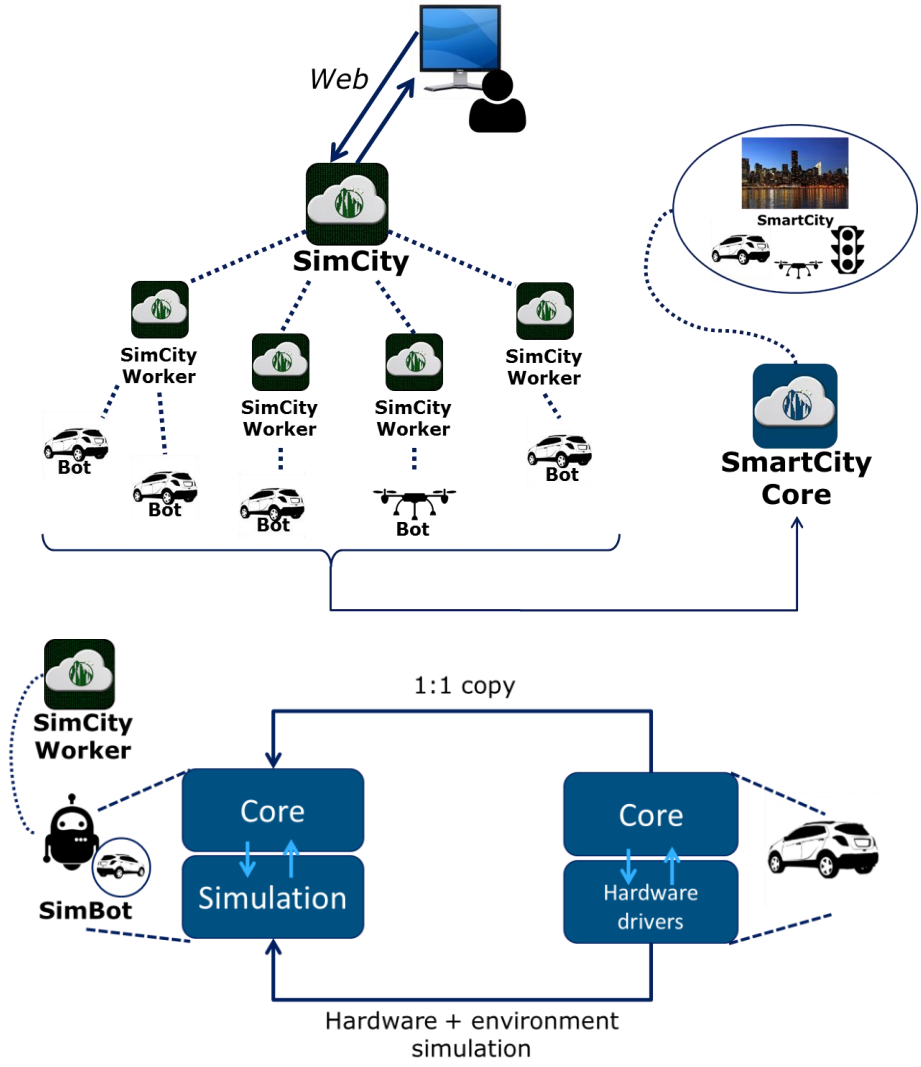
3 SimCity Modules

3.1 SimCity

De SimCity module is een webservice die een web front-end aanbied om de status van SimCity Workers op te vragen in een overzichtelijke gebruiksvriendelijke dashboard. Daarnaast zal deze de mogelijkheid aanbieden om nieuwe bots te instantiëren en controleren. In de huidige implementatie van dit project is de Central module nog niet functioneel geïmplementeerd.

3.2 SimCity Worker

De SimCity Worker module staat in voor de simulatie van bots in de SmartCity. Voor elke simulatie zijn er twee onderdelen: de software core (Java) en de gesimuleerde driver voor de hardware. De core is een identieke kopie van de core die wordt gebruikt door de SmartBots in de SmartCity. Voor de simulatie zal naast een core, een simulatie laag opgestart die de actuatoren en sensoren simuleert. Om een realistische simulatie te bekomen, wordt de locatie en omgeving gesimuleerd om overeenkomstige sensor info te verkrijgen in de simulatie. De SimCity Worker zal meerdere bots parallel kunnen lopen. Hierbij verzorgt het systeem dat de simulatie laag wordt gekoppeld aan de juiste bot core.

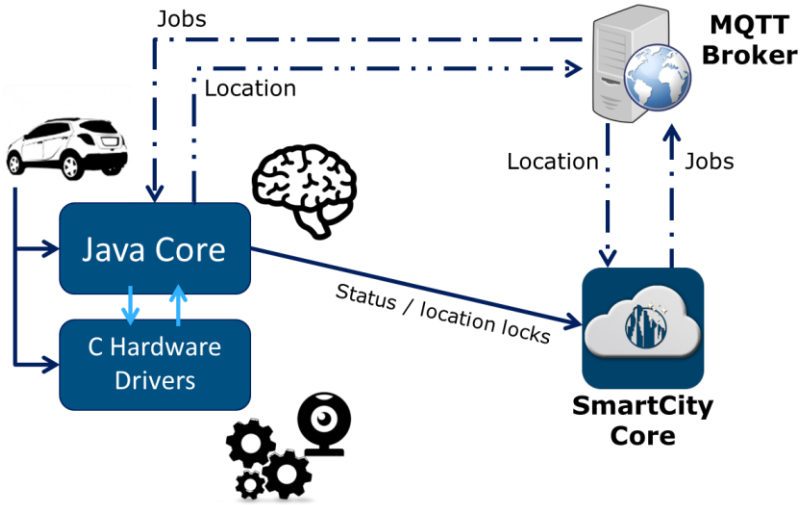


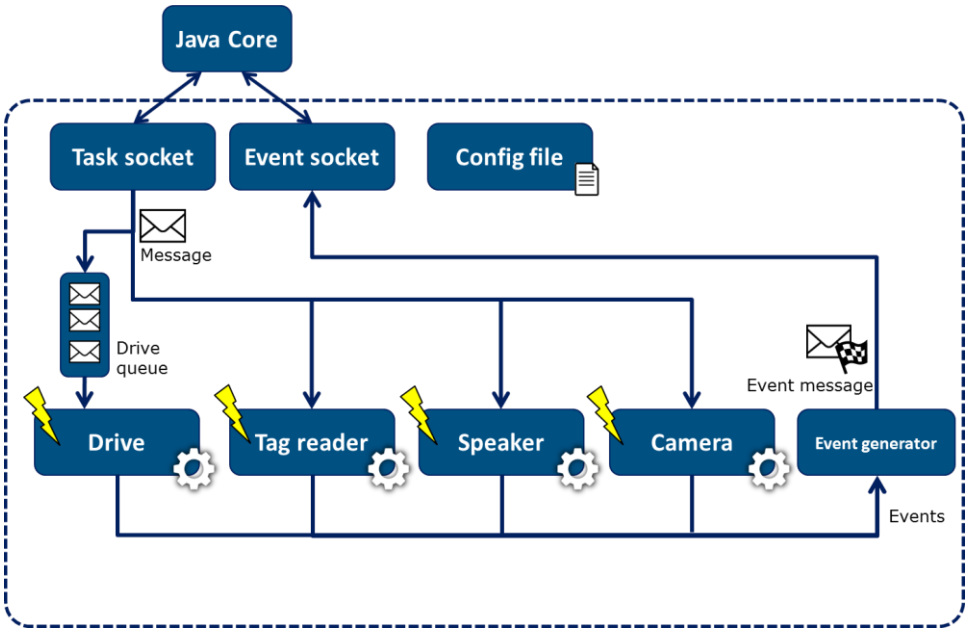
4 SmartBots

4.1 SmartCar

Een eerste type van bots in de SmartCity is de SmartCar. Dit tweewielig gemotoriseerd voertuig is uitgerust verschillende sensoren, zoals camera, tag reader, speaker, vorklift,… Voor de implementatie bestaat de software uit twee software lagen: Core (java) en hardware drivers (C). De high level core is het brein van de SmartCar die communiceert met de SmartCity Core. De communicatie wordt voorzien via een REST interface om punt locks aan te vragen om op een kruispunt te rijden. De locatie updates en jobs worden verzonden via MQTT.

De low-level driver laag biedt een interface aan de Java Core. Via twee sockets kunnen taken gestuurd worden en systeem events ontvangen worden door respectievelijk de 'Task' en 'Event' socket. Het event systeem wordt enkel geactiveerd indien een client is verbonden met de socket. Voor elke gestuurde taak wordt een module thread gestart die toelaat om de taken asynchroon uit te voeren zodanig dat de SmartCar meerdere functies simultaan kan uitvoeren of onderbreken. Het management systeem houd hierbij rekening dat er geen resource conflicten kunnen ontstaan door twee taken die dezelfde resources aanspreken. Voor de drive module is een queue geïmplementeerd die toelaat om meerdere rij instructies door te geven waarna deze één na één worden uitgevoerd. De inhoud van deze queue kan ten alle tijden geflushed worden indien nodig. In de instructionset onderaan kunnen de andere geïmplementeerde functies geraadpleegd worden.





Known commands for SMARTCAR system (v. 0.0.3 - 18/05)

```
DRIVE COMMANDS
==============

DRIVE ABORT : Abort the running drive command and flush all queued drive commands
DRIVE FLUSH : Flush all queued drive commands (Does not interrupt running drive command)
DRIVE PAUSE : Pause the running drive command
DRIVE RESUME : Unpause / resume the running drive command
DRIVE FOLLOWLINE : Follow roadline with lightsensors until end of line is reached. Returns
when action is finished with event (DRIVE EVENT: FINISHED)
DRIVE FOLLOWLINE [x] : Follow roadline with lightsensors until a distance of x is reached
(does not stop on line end). Returns when action is finished with event (DRIVE EVENT:
FINISHED)
DRIVE FORWARD [x] : Drive straight forward until a distance of x is reached. Returns when
action is finished with event (DRIVE EVENT: FINISHED)
DRIVE BACKWARDS [x] : Drive straight backwards until a distance of x is reached. Returns
when action is finished with event (DRIVE EVENT: FINISHED)
DRIVE TURN [L / R] : Make a 90° turn to the left / right. Returns when action is finished
with event (DRIVE EVENT: FINISHED)
DRIVE TURN [L / R] [x] : Make a x° turn to the left / right. Returns when action is
finished with event (DRIVE EVENT: FINISHED)
DRIVE ROTATE [L / R] [x] : Rotate stationary counter-clockwise / clockwise with a angle of
x°. Returns when action is finished with event (DRIVE EVENT: FINISHED)
DRIVE DISTANCE : Read the wheel encoders and returns x, the average travelled distance in
mm with event (TRAVEL DISTANCE EVENT: [x])


CAMERA COMMANDS
===============

CAMERA TRAFFICLIGHT : Use front-camera to detect state of the traffic light. Returns state
with event (TRAFFICLIGHT DETECTION EVENT: [GREEN, RED, NONE, ERROR])


TAGREADER COMMANDS
==================

TAG READ UID : Read a tag with the tagread. Returns the UID of the tag with event (TAG
DETECTION EVENT: [$ID, NONE])
```

```
LIFT COMMANDS
=============

LIFT GOTO [x] : Move the lift to x mm absolute height. Return when action is finished with
event (LIFT GOTO EVENT: DONE)
LIFT HEIGHT : Get the position of the lift and returns x, the height of the fork in mm with
event (LIFT HEIGHT EVENT: [x])


SPEAKER COMMANDS
================

SPEAKER MUTE : Mute all sounds and voices from the speaker
SPEAKER UNMUTE : New sounds and voices will play on the speaker
SPEAKER STOP : Stop the current playing sound or voice
SPEAKER PLAY [x] : Play WAV file in the audio folder with the filename x.wav
SPEAKER SAY [x] : Use the voice synthesizer to say the message x


OTHER COMMANDS
==============

SHUTDOWN : Turn the SMARTCAR system off
HELP / ? : Get a list of all known commands
```

4.2 SmartTrafficLight

Een tweede fysieke bot is de Smart Traffic Light. Deze bot is een verkeerslicht die voertuig bots dienen te respecteren. De SmartCar heeft een camera waarmee het de status van het licht kan bekijken. Daarnaast is het de bedoeling dat de status van het licht wordt doorgegeven aan de SmartCity Core. Zo ook kan de SmartCity Core de status kunnen wijzigen om zo prioritaire voertuigen door te laten. In de huidige implementatie van dit project is de Smart TrafficLight module nog niet functioneel geïmplementeerd.

5 Project Features

5.1 Version 0.3 - APLHA

- SmartCity Project
  - Spring framework
    - Security
    - IoC container
    - MVC pattern (voor front-end modules)
  - Maven (automatic dependencies resolving)
  - Eenvoudig switchen naar verschillende run configuraties
    - Default
    - Developers modus
      - Testen op een in-memory database.
    - Standalone
      - Start eigen embedded Tomcat instantie op.
  - Meerdere talen ondersteuning van web front-ends
  - Bootstrap (web lay-out)
  - Thymeleaf (server gerenderde webpagina's → beveiliging en afscherming van code)
  - Stateless server modules → horizontale scaling van services mogelijk
  - Web front-ends compatibel met vele diverse apparaten met een compatibele webbrowser. Lay-out wordt aangepast naargelang het toestel: PC, tablet of smartphone. → geen software installatie nodig.
- SmartCity Core
  - Semafoor blokken van een kruispunt om botsingen te voorkomen.
    SmartCity Core > PointController.java
    SmartCar Core > QueueConsumer.java > run()
  - Locatie updates over MQTT.
  - Jobs zenden naar bots over MQTT.
    SmartCity Core > ./controllers/mqtt
    SmartCity Car > ./controllers/mqtt
  - Status updates van bots over REST interfaces:
    - SmartCars
    - SmartTrafficLights (planned feature)
  - Terminal functionaliteit voor het beheren van de Core en het sturen van jobs naar de SmartBots.
    SmartCity Core > ./services/TerminalService.java
    SmartCity Core > ./tools/Terminal.java
- SmartCity Viewer
  - Dynamisch opbouwen van een map voor visualisatie. Opmaak van de map wordt berekend uit de lengte van de wegen en hun onderlinge connecties.
    SmartCity Viewer > ./tools/MapBuilder.java

    - Web front-end met een view van de map met waarop de bots in de SmartCity worden aangeduid.
        - HTML Canvas
    - Locaties van de bots worden live geüpdatet op de Viewer.
- SimCity Worker
    - Instantiëren en beheren van SimBots.
    - Meerdere simulaties kunne parallel uitgevoerd worden.
    - Een instantie van de Java Core van een SmartBot wordt uitgevoerd bij een instantie zodanig dat de simulatie steeds accuraat is met de laatste versie op de SmartBots zelf.
    - De Bot Core wordt geconfigureerd door middel van een configuratie bestand (BotCoreConfig.xml).
    - De hardware simulatie laag simuleert de werking van de low-level drivers. De sensor waardes van de gesimuleerde laag worden zo realistisch mogelijk benaderd door de omgeving van de bot te simuleren door extra informatie van de SmartCity Core op te vragen (pad lengte, map, richtingen, RF-tags,…).
    - Automatische Bot Core configuratie om deze te koppelen met de overeenkomstige simulatie laag.
    - Bot Core logger functionaliteit.
- SmartCar Core
    - TCP communicatie met hardware driver (sturen van commando's en ontvangen van status events).
    - REST communicatie met SmartCity Core (opvragen van map, point locks aanvragen).
    - MQTT implementatie voor locatie updates.
    - MQTT implementatie voor ontvangen van jobs.
    - Commando queue naar hardware voor het opstapelen van commando's tot de hardware klaar is voor uitvoer.
    - Padplanning functionaliteit
        - Dijkstra
        - Random
    - Point locking aanvragen op server om de beschikbaarheid van een kruispunt te checken.
- SmartCar Driver
    - SmartWatcher: starten/stoppen van systeem met reset knop achteraan.
    - Asynchroon taken afhandeling.
    - Drive module met diverse commando's.
    - Onderbreken van drive commando's
    - Drive queue voor het doorsturen van een batch aan drive commando's.

- Opvragen van sensorinformatie via event systeem: taak (request voor info) → start proces (sensor meting) → resultaat verzonden als event
  - ➔ Taken listener wordt hierdoor niet geblokkeerd indien het opvragen, meten van sensor informatie meer tijd vraagt.
- Configuratie bestand.
- Ondersteuning voor sensor modules en diverse: lift, tag reader, camera, speaker (zie instructionset op pagina 13).
- Audio player.
- Verkeerslichten detectie.
- Extra hidden features: QR-code reader, dijkstra padplanning, json parser, rest-interface, watchdog…).

En nog vele kleine features die de SmartCity project nog beter maken!

6 Future Work

- Implementatie van de verschillende geplande web front-ends voor elke SmartCity module.
- Verkeerslichten (Smart TrafficLights) integreren in SmartCity (bediening en controleren van city flow).
- Powercontrol in SmartCar: automatisch naar parkeerplaats begeven voor opladen.
- Mobile app
- City map bouwen/instellen met gebruiksvriendelijke web interface ipv handmatig entries plaatsen in database.
- User management: gebruikers definiëren met specifieke privileges (bots besturen, verkeerslichten wijzigen,…)
- Systeem optimalisatie en performantie verbeteringen.
- Drones.
- Traffic control en flow.
- Geprioriteerde voertuigen introduceren.
- Data analysis op big data van SmartCity (verkeer,…) en bijhorende automatische acties ondernemen.
- Grote schaal simulaties uitvoeren.
- …

En vele andere ideeën voor een betere SmartCity!