TIANE

Guide zur Modulentwicklung

FERDINAND und KLARA KRÄMER

Bundeswettbewerb Jugend Forscht 2019

Mathematik/Informatik

Einer der großen Vorteile des TIANE-Systems ist die sehr einfache und doch sehr umfangreiche Erweiterbarkeit durch eigene Module. Damit du diesen Vorteil auch nutzen kannst, soll dir dieser Guide einen Einblick in die Modulentwicklung für TIANE bieten – von einem einfachen Beispiel für Einsteiger bis hin zu den tief ins System greifenden Möglichkeiten für Fortgeschrittene.

TIANE bringt von Haus aus bereits viele nützliche Funktionen moderner Sprachassistenten mit: Einkaufslisten, Timer, Wecker, Wettervorhersage, eine Wikipedia-Suche, unterhaltsame Kommentare, Zitate und vieles mehr.
Trotzdem wirst du eventuell irgendwann feststellen, dass genau deine Wunschfunktion fehlt, seien es deine eigenen Insiderwitze, die Ansteuerung exotischer Smart-Home-Geräte oder auch nur eine personalisierte Morgenroutine. Was also tun? Auf einen anderen, womöglich kommerziellen Sprachassistenten umsteigen, weil diese vielleicht mehr Funktionen bieten? Eine Mail mit dem Funktionswunsch an das TIANE-Team schreiben und eventuell lange auf eine Antwort oder gar Lösung warten? Viel besser: Dank der modularen Erweiterbarkeit von TIANE kannst du deine Idee selbst programmieren und vielleicht sogar anderen zur Verfügung stellen! Und da Module – wie übrigens auch das gesamte TIANE-System – in der einsteigerfreundlichen Sprache Python programmiert werden, kannst du das ganz einfach lernen (solltest du vorher noch einen kleinen Python-Grundkurs benötigen, können wir den kostenlosen, interaktiven Kurs auf codecademy.com sehr empfehlen; auf diese Weise hat auch das gesamte TIANE-Team programmieren gelernt :-)).

Dieser Guide bietet dir zunächst einen einfachen Einstieg anhand des Beispiels eines Moduls, mit dem TIANE auf Kommando „einen Würfel wirft“, dir also eine zufällige Zahl zwischen 1 und 6 ausgibt, und zeigt dir dann die Möglichkeiten zur Entwicklung eigener, komplexerer Module.

Vorüberlegungen

Bevor du dein Modul programmieren kannst, musst du dir erst einmal überlegen, was es überhaupt können soll, und dich auf Basis dessen für eine bestimmte Art von Modul (oder eine Verkettung von mehreren Modulen verschiedener Arten) entscheiden. Je nachdem, welche Art du wählst, stehen dir unterschiedliche Funktionen zur Verfügung und du musst das fertige Modul zum Starten an unterschiedlichen Orten ablegen. Klingt unnötig kompliziert? Nun, unnötig ist es leider nicht, sonst hätten wir das sicher anders gelöst. Aber mit den hier gegebenen Hilfestellungen ist es zum Glück auch nicht wirklich kompliziert.

Warum man im TIANE-System zwischen verschiedenen Arten von Modulen unterscheidet, wird schnell klar, wenn du die große Bandbreite an Möglichkeiten betrachtest, die sich mit Modulen abdecken lässt:

- Manche Module reagieren nur auf Zuruf des Nutzers, zum Beispiel „Hey Tiane, wirf einen Würfel“
- Manche Module laufen immer im Hintergrund und überprüfen zum Beispiel regelmäßig deine Mails, um dir die neuen Nachrichten vorzulesen

- Manche Module benötigen direkten Zugriff auf die TIANE-Hardware (Raspberry Pi o.ä.) in einem bestimmten Raum, um zum Beispiel eigene Smart-Home-Funktionen steuern zu können
- Manche Module benötigen Zugriff auf die leistungsfähigere Hardware deines TIANE-Servers, um zum Beispiel eine Gesichtserkennung laufen zu lassen
- Und vielleicht sollen manche dieser Module ihre Funktionen nicht jedem im Haus zur Verfügung stellen, sondern dein ganz persönlicher „Geheimtrick“ sein

Daraus ergeben sich folgende Modularten:

Module, die auf der TIANE-Hardware in einem bestimmten Raum laufen:

- **room_modules**
  Room_modules starten auf Kommando des Nutzers (oder Aufruf von einem anderen Modul aus) und laufen direkt auf dem TIANE-RasPi im Zielraum, haben also vollen Zugriff auf die Hardware (USB, GPIO, WLAN, Bluetooth,…) dieses RasPis und sind somit perfekt geeignet für Smart-Home-Lösungen wie Lichtschalter oder Rollläden dieses Raumes. Gut zu wissen: Dank der TIANE-Netzwerkfunktionen können Module, die lokal in einem Raum laufen, trotzdem Module in beliebigen anderen Räumen oder auf dem Server aufrufen, TIANE in anderen Räumen reden oder zuhören lassen und Daten mit anderen TIANE-Geräten austauschen und umgekehrt (Details dazu weiter unten). Du kannst sogar per Sprachkommando room_modules in einem völlig anderen Raum aufrufen, indem du einfach den Namen des Zielraums in das Kommando einbaust: So funktioniert zum Beispiel „Hey Tiane, mach das Licht im Wohnzimmer an“ auch, wenn du dich in der Küche befindest. Fertige room_modules müssen zum Ausführen im Ordner „modules“ im TIANE-Ordner des entsprechenden Raums abgelegt werden.
- **continuous_room_modules**
  Continuous_modules sind, wie der Name schon sagt, die guten Geister, die immer im Hintergrund laufen; meistens verwendet, um bestimmte Werte zu überwachen und darauf selbst zu reagieren oder den Nutzer zu benachrichtigen. Das spezifische Aufgabengebiet von continuous_room_modules ist demnach vor allem die stetige Überwachung und Regelung von Smart-Home-Geräten in diesem Raum, die auch ohne Kommandos des Nutzers funktionieren sollen, zum Beispiel Heizungssteuerungen oder automatische Licht- oder Rollladenschaltungen.
  Continuous_modules werden von TIANE automatisch ungefähr in für jedes Modul individuell festlegbaren Zeitabständen aufgerufen, von wenigen Millisekunden bis hin zu beliebig langen Zeiträumen. Wichtig: Continuous_modules können im TIANE-Universum nicht direkt zum Nutzer sprechen oder ihm zuhören und sollten generell für ihre einmalige Ausführung möglichst wenig Zeit brauchen, da alle continuous_modules nacheinander verarbeitet werden und somit andere in der Reihe eventuell warten müssten. Zur Kommunikation mit dem Nutzer können continuous_modules einfach kleine „normale“ Module starten, die genau für diese Aufgabe besonders leicht zu programmieren sind. Damit sie geladen und ausgeführt werden können, müssen continuous_room_modules im Ordner „modules/continuous“ im TIANE-Ordner des Raumes abgelegt werden.

Module, die auf dem hauseigenen TIANE-Server laufen:

Module, die alle Nutzer betreffen:

- **common_modules**
  Common_modules werden wie room_modules per Sprachkommando aufgerufen, stehen aber automatisch in jedem Raum zur Verfügung. Dazu laufen sie auf deinem TIANE-Server, zu dem jeder TIANE-Raumclient die bei ihm eingegangenen Sprachbefehle sendet. Sie sind, wie der Name schon andeutet, perfekt geeignet für allgemeine Aufgaben eines Sprachassistenten: Wettervorhersage, Timerfunktionen und die Einkaufsliste der Familie sollen schließlich ohne Probleme von jedem Raum aus aufrufbar sein und trotzdem, wie zum Beispiel im Falle der Einkaufsliste, immer auf dieselben Daten zugreifen können. Da die Module auf dem Server liegen, haben sie aber keinen direkten Zugriff auf die Hardware der einzelnen Raumclients, sie können auf diesen nur allgemeine TIANE-Funktionen ausführen (sprechen, auf Antworten hören, ein Modul in diesem Raum starten). Common_modules müssen im Ordner „modules“ im TIANE-Ordner deines TIANE-Heimservers (oder Hauptrechners) abgelegt werden.
- **continuous_common_modules**
  Continuous_common_modules werden genauso wie continuous_room_modules in einstellbaren, regelmäßigen Zeitabständen immer wieder automatisch ausgeführt, nur eben auf dem Server. Dadurch haben sie wie normale common_modules keinen Zugriff auf die Hardware eines konkreten Raumes, dafür aber auf die meist leistungsfähigere des Servers, denn das ist die Hauptaufgabe dieser Art von Modulen: leistungshungrige Prozesse zentral auszuführen. Wusstest du zum Beispiel, dass die gesamte TIANE-interne Gesichtserkennung, die dafür sorgt, dass TIANE stets weiß, wo sie jeden Nutzer findet, nur als continuous_common_module angebunden ist (und die dafür benötigten Kamerabilder über die TIANE-Netzwerkfunktionen von contiunuous_room_modules zugeschickt bekommt)? Dieses Beispiel zeigt auch sehr gut die Möglichkeiten, mit Modulen, insbesondere solchen auf dem Server, tief in das TIANE-System einzugreifen (mehr dazu später). Natürlich kommen continuous_common_modules auch für weniger aufwendige Aufgaben zum Einsatz, wie zum Beispiel die Verwaltung des Familienkalenders, der regelmäßig überprüfen muss, ob es Zeit ist, einen Nutzer über einen anstehenden Termin zu benachrichtigen. Der Speicherort für continuous_common_modules ist der Ordner „modules/continuous“ im TIANE-Ordner des Servers.

Module, die nur einzelne Nutzer aufrufen können

- **user_modules**
  User_modules sind ein ganz besonderer Trick von TIANE, der nur dadurch möglich wird, dass TIANE dank ihrer Stimmen- und Gesichtserkennung stets weiß, welcher Nutzer gerade zu ihr spricht, und ermöglichen individuelle Funktionen für jeden Nutzer! Jeder Nutzer kann seine user_modules (die natürlich auf dem Server liegen und per Sprachbefehl in jedem Raum aufgerufen werden können; ich denke, du hast das Schema inzwischen verstanden) in den modules-Ordner seines Nutzerverzeichnisses im TIANE-Ordner auf dem Server (also „users/NameDesNutzers/modules“) legen und sich so eigene, exklusive Funktionen schaffen (wie zum Beispiel kleine Witzeleien mit TIANE oder eine ganz persönliche Morgenroutine, bei der TIANE dich nicht nur weckt, sondern auch gleich die Kaffeemaschine einschaltet) und sogar vorhandene, allen zur Verfügung stehende Funktionen durch eigene „überschreiben“. Bei Erhalt eines Sprachkommandos durchsucht TIANE nämlich zuerst die user_modules des betreffenden Nutzers, und nur wenn sie dort nicht fündig wird auch die common- und room_modules. So kannst du zum Beispiel nur für dich ein individuelles Begrüßungs-Modul programmieren, während TIANE allen anderen im Haus immer noch auf die gleiche Art „Hallo“ sagt.
- **continuous_user_modules**
  Wie du wahrscheinlich schon geahnt hast, gibt es auch die user_modules nochmal in der fortlaufenden Variante. Continuous_user_modules könnten zum Beispiel zur Verwaltung deines persönlichen Kalenders dienen, deine Mails checken oder Ähnliches. Sie müssen im Ordner „users/NameDesNutzers/modules/continuous“ liegen.

Die geeignetste Art für unser Würfel-Modul ist demnach ein „common_module“: Das Modul soll per Sprachbefehl aufgerufen werden, ohne Angabe eines Raumes überall im Haus verfügbar sein und allen Nutzern zur Verfügung stehen (es sei denn, du willst es ganz für dich allein haben, in dem Fall wäre ein „user_module“ eine gute Wahl).

Programmierung deines Moduls

Nachdem du dich anhand der Aufzählung oben für eine Art von Modul (oder eine Verkettung von Modulen, dazu später noch mehr) entschieden hast, musst du es nur noch programmieren. Dafür müssen wir allerdings zunächst mal kurz auf die Art und Weise eingehen, wie TIANE überhaupt mit (vom Nutzer gestarteten; continuous_modules kommen später) Modulen arbeitet.

Wenn ein Nutzer in einem Raum einen Sprachbefehl gibt, in unserem Fall also zum Beispiel so etwas wie „Hey Tiane, wirf einen Würfel“, geht TIANE zuerst alle Module grob durch, um festzustellen, welches Modul mit diesem Befehl etwas anfangen kann, und zwar in der Reihenfolge

1. user_modules des Nutzers
2. common_modules

3. falls im Kommando ein bestimmter Raum angegeben war: room_modules des Zielraumes
4. ansonsten: room_modules des Raumes, in dem der Nutzer den Befehl gegeben hat

Da TIANE ja nicht wissen kann, was dein neues Modul kann, müssen Module bei diesem „groben Durchgehen“ selbst entscheiden, ob sie sich für das jeweilige Kommando zuständig fühlen. Dafür braucht jedes Modul eine Funktion namens `isValid()`, die als Argument den Text des gegebenen Kommandos (allerdings OHNE das Aktivierungswort, also den „Hey Tiane“-Part!) als String überreicht bekommt und auf Basis dessen `True` („Ja, ich kann dieses Kommando verstehen und ausführen“) oder `False` („Nö, der Text sagt mir nichts, soll sich das nächste Modul drum kümmern“) ausgeben muss. Für unser Würfelmodul könnte diese Funktion zum Beispiel so aussehen:

```
def isValid(text):
    if 'würfel' in text.lower():
        return True
    else:
        return False
```

Diese Funktion überprüft einfach nur, ob das Schlüsselwort „würfel“ in dem Kommando vorkommt. Da das zum Glück ein recht seltenes Wort ist, reicht diese Variante völlig, und da wir nur nach diesem einen Schlüsselwort suchen, reagiert die Funktion nicht nur auf „Wirf einen Würfel“, sondern zum Beispiel auch auf „Würfel mal kurz“ – also sehr praktisch. Achte dabei aber auf ein paar Dinge: Manche, insbesondere ältere, stt (Speech-to-text)-Engines (Spracherkennungsprogramme) geben das gesamte erkannte Kommando in Kleinbuchstaben aus, während modernere natürlich Nomen wie „Würfel“ groß schreiben. Für optimale Kompatibilität bietet es sich daher oft an, einfach mit `text.lower()` den gesamten Text in Kleinbuchstaben zu verwandeln, damit du nicht nach zwei verschiedenen Schreibweisen deiner Schlüsselwörter suchen musst. Außerdem zeigt dieses Beispiel nur den einfachsten Fall. Falls deine Schlüsselwörter auch als Teil von anderen Wörtern auftauchen können (z.B. suchst du nach „an“ und „der“, und das Modul springt auf den Satz „Mein Name ist Alex**ander**“ an…), musst du ggf. die geforderten Leerzeichen mit einbeziehen, oder am besten gleich Pythons eingebaute „regex“ (regular expressions)-Funktionen verwenden. Auch kann es manchmal hilfreich sein, direkt eine etwas umfangreichere Logik einzubauen, die z.B. nur `True` ausgibt, wenn „dieses oder jenes und jenes, aber nicht dieses andere“ Wort vorkommt, und so weiter.

Wo es sich nicht vermeiden lässt, dass zwei Module nach denselben Schlüsselwörtern suchen, aber eines von beiden noch ein anderes Kriterium hat, bietet die TIANE-Modulschnittstelle übrigens noch eine nützliche Hilfe an: Du kannst mit dem Hyperparameter PRIORITY die Module so sortieren, dass das Modul mit der zusätzlichen Abfrage als erstes aufgerufen wird, sodass das andere nur zum Tragen kommt, wenn das erste nichts mit dem Kommando anfangen kann. PRIORITY kann beliebige positive Zahlen erhalten, die Module werden dann vor Abfrage der `isValid()`-Funktionen nach dieser Zahl sortiert, sodass Module mit höherer Priorität zuerst abgefragt werden. Die PRIORITY deines Moduls definierst du einfach mit einer Zeile am Anfang deines Quelltextes:

```
PRIORITY = 10
```

Für Module, die keinen PRIORITY-Parameter setzen, wird automatisch eine Priorität von 0 angenommen. Bitte beachte, dass die oben genannte Reihenfolge user_modules → common_modules → room_modules von der PRIORITY unbeeinflusst bleibt, diese sortiert die Module nur innerhalb ihrer Kategorien.

Nachdem TIANE nun die `isValid()`-Funktionen der Module der Reihe nach aufgerufen und eine davon `True` ausgegeben hat, geht TIANE davon aus, dass das Kommando für dieses Modul gedacht war, muss also das Modul ausführen, damit es das Kommando verarbeiten kann. Dafür braucht dein Modul eine Funktion namens `handle()`, die TIANE nun aufruft und die folgende Parameter akzeptiert:

- text
  Der Text des Kommandos, derselbe, den auch `isValid()` bekommen hat (also wieder ohne „Hey Tiane“)

- tiane
  Das ist die Klasse, mit der TIANE deinem Modul all ihre Möglichkeiten zur Verfügung stellt: Hier finden sich Funktionen um Nutzern per Sprachausgabe zu antworten, per Spracherkennung auf weitere Eingaben zu hören, mit anderen Räumen oder dem Server zu kommunizieren, andere Module zu starten, einen Satz des Nutzers analysieren zu lassen und vieles mehr. Eine genauere Dokumentation aller Funktionen des `tiane`-Objekts folgt weiter unten.

- local_storage
  Der `local_storage` ist ein Dictionary, das für alle Module (normale wie fortlaufende) auf einem TIANE-Gerät gleich ist. Hier können sie Daten untereinander austauschen oder einfach nur für sich selbst speichern und wieder abrufen. `local_storage` enthält schon zu Beginn einige Felder, die Daten über Räume und Nutzer in deinem TIANE-Netzwerk zur Verfügung stellen, eine genauere Dokumentation folgt weiter unten.

Für unser „Würfeln“-Modul könnte die handle-Funktion zum Beispiel so aussehen:

```
from random import randint

def handle(text,tiane,local_storage):
    würfelzahl = randint(1,6)
    tiane.say('Ich habe eine {} für dich gewürfelt'
              .format(str(würfelzahl)))
```

Hier wird einfach mit der Python-eigenen Zufallsfunktion eine Zahl zwischen 1 und 6 generiert und das Ergebnis als ganzer Satz über die Funktion `tiane.say()` ausgegeben. Diese `say()`-Funktion ruft TIANEs (frei wählbare) Standard-Sprachausgabe auf, sodass TIANE immer mit der gleichen Stimme spricht, was ja auch gewollt ist. Neben dem zu sprechenden Text akzeptiert `tiane.say()` auf Wunsch übrigens auch noch andere sehr interessante Parameter, dazu später mehr.

Damit ist unser Beispielmodul fertig, du hast jetzt deiner TIANE beigebracht, für dich zu würfeln und dir das Ergebnis mitzuteilen. Um das Modul zu laden, musst du es jetzt nur noch in den entsprechenden Ordner („modules“) auf deinem TIANE-Server legen und entweder das Programm neu starten oder mit dem Sprachbefehl „lade die Module neu“ einbinden, danach wird dein Modul automatisch bei jedem Sprachbefehl mit berücksichtigt. Dieses Modul ist natürlich nur ein sehr einfaches Beispiel, das nur einen Bruchteil der Möglichkeiten der Modulschnittstelle ausnutzt, aber für viele lustige und nützliche Ideen bereits ausreicht. Und für alles andere liest du einfach hier weiter…

Das WORDS-System – eine Hilfestellung für alte Spracherkennungssoftware

Wie schon erwähnt, soll TIANE möglichst auch mit älterer Offline-Spracherkennungssoftware wie z.B. PocketSphinx kompatibel sein. Um halbwegs vernünftig zu funktionieren, benötigen diese aber teilweise einen kleinen „Spickzettel“: Eine Liste von Wörtern, die für den Betrieb wichtig sind und somit im Zweifelsfall eher erkannt werden, sollte die Spracherkennung sich an einer Stelle unsicher sein. Daher hast du die Möglichkeit, in deinen TIANE-Modulen eine solche Liste von Wörtern anzulegen, die von diesem Modul benötigt werden und daher bevorzugt erkannt werden sollen. Dies geschieht über den zusätzlichen Hyperparameter WORDS, der für unser eben gezeigtes Würfelmodul zum Beispiel so aussehen könnte:

```
WORDS = ['Würfel', 'würfeln']
```

Die WORDS-Listen aller Module im TIANE-Netzwerk werden gesammelt und den Speech-to-Text-Programmen zur Verfügung gestellt. Aufgrund der besseren Kompatibilität kann es als guter Stil betrachtet werden, jedes Modul mit einem WORDS-Parameter auszustatten, der alle von diesem Modul eventuell benötigten Wörter enthält; da wir in unseren Testaufbauten aber keine solche alte Spracherkennung verwenden, die davon profitieren würde, haben wir uns ehrlich gesagt selbst nicht immer an diese Empfehlung gehalten. Die Möglichkeit soll hier trotzdem nicht unerwähnt bleiben.

Telegram

Das TIANE-System bringt von Haus aus eine gute Anbindung an die Messenger-App Telegram mit. So kannst du jederzeit von unterwegs mit deinem Sprachassistenten kommunizieren, als wärst du zu Hause, und zwar nicht nur schriftlich, sondern auch ganz einfach per Sprachnachricht. Wenn du diese Telegram-Anbindung bei der Einrichtung auswählst, funktioniert sie automatisch mit allen deinen Modulen: Schreibst du TIANE eine Nachricht, wird sie mit dem Text der Nachricht (oder im Falle einer Sprachnachricht automatisch mit dem erkannten Text) die `isValid()`-Funktionen der Module durchgehen, und wenn ein Modul via Telegram aufgerufen wurde, nutzen auch sämtliche `say()`- und `listen()`-Funktionen dieses Moduls ganz intuitiv deinen Telegram-Chat als Ein- und Ausgabe.

Gleichzeitig gibt dir die Modulschnittstelle natürlich auch die Möglichkeit, die spannenden Funktionen der Telegram-Anbindung jenseits dieser einfachen automatischen Textkommunikation zu nutzen: So kannst du beispielsweise beim Aufruf von `say()` oder `listen()` mit einem zusätzlichen Argument (siehe unten) festlegen, dass diese Ein- oder Ausgabe immer auf eine bestimmte Weise erfolgen soll (also immer per Telegram oder immer per Sprache), im Falle von `say()` kannst du TIANE sogar anweisen, dir den Text

ihrerseits als Sprachnachricht zu schicken. Außerdem können via Telegram auch ganz andere Medien außer Nachrichten versandt werden, zum Beispiel Bilder, Videos, Dokumente, Audiodateien usw., die du vielleicht auch mit Modulen verarbeiten möchtest. Dafür kannst du in Modulen die zusätzliche Funktion `telegram_isValid()` definieren, die grundsätzlich die gleiche Aufgabe hat, wie die normale `isValid()`-Funktion, mit dem Unterschied, dass sie nur bei Anfragen via Telegram aufgerufen wird und als Argument keinen Text erhält, sondern ein (leicht erweitertes) telepot-Message-Dictionary (siehe https://telepot.readthedocs.io/en/latest/#receive-messages und die restliche telepot-Dokumentation für Details). Module dürfen sowohl eine `isValid()`- und eine `telegram_isValid()`-Funktion als auch nur eine von beiden besitzen. Die Reihenfolge user_modules → common_modules → room_modules gilt auch für Telegram-Aufrufe, wobei room_modules natürlich nur zum Tragen kommen, wenn im Text der Nachricht ein TIANE-Raum erwähnt wird. Bei einem Aufruf via Telegram werden zuerst die `telegram_isValid()`-Funktionen aller Module einer Gruppe abgefragt, dann erst die „normalen“. Für die `handle()`-Funktion steht das telepot-Message-Dictionary der ursprünglichen Nachricht unter `tiane.telegram_data` zur Verfügung (auch, wenn das Modul keine spezielle `telegram_isValid()` Funktion besitzt). Wenn du das telepot-Message-Dictionary von späteren Antworten des Nutzers erhalten möchtest, nutze dafür `tiane.telegram_listen()` statt `tiane.listen()`.

Natürlich kannst du deine TIANE-Module auch auf gleichem Wege andere Medien an den Nutzer zurückschicken lassen: Dafür nutzt du am besten die ganz normalen telepot-send-Funktionen (https://telepot.readthedocs.io/en/latest/reference.html#telepot.Bot.sendMessage), die telepot-Bot-Instanz erreichst du im TIANE-System unter `tiane.telegram.bot`. Die Telegram-Chat-IDs für die einzelnen TIANE-Nutzer erhältst du am einfachsten im Local_storage (`local_storage[,TIANE_telegram_name_to_id_table‘][,NameDesNutzers‘]`).

Einführung in continuous_modules

Unser Beispielmodul von eben kann nur auf Zuruf aktiviert werden, mit einem Satz antworten und sich dann wieder beenden. Als wirklich „smarter“ Heimassistent soll TIANE aber auch selbst aktiv werden und dich ohne vorherige Aufforderung ansprechen können, um dich zum Beispiel an Termine zu erinnern. Oder irgendetwas anderes, wie zum Beispiel eine Gesichtserkennung, im Hintergrund ausführen. Dafür gibt es im TIANE-Universum die große Gruppe der continuous_modules.

Aber warum braucht man dafür eigentlich eine eigene Art von Modulen? Reicht es nicht einfach, in ein normales Modul eine Schleife einzubauen, sodass es sich eben nicht sofort wieder beendet? Prinzipiell geht das tatsächlich, eine solche Konstruktion stellt eigentlich sogar noch eine ganz andere Art von Modulen dar. Allerdings ist diese Vorgehensweise wirklich nur in absoluten Ausnahmefällen empfohlen, in denen es aus unerfindlichen Gründen essentiell ist, dass das Modul tatsächlich in einem eigenen Thread im Hintergrund läuft. Denn genau das ist der Unterschied zwischen normalen- und continuous_modules: Bei jedem Start eines normalen Moduls, ob durch Sprachbefehl oder Aufruf aus einem anderen Modul heraus, wird ein neuer Thread für dieses Modul gestartet, während continuous_modules sehr ressourcenschonend alle in einem Thread nacheinander ausgeführt werden. Falls deiner TIANE-Hardware also die Belastung durch viele ständig

laufende Threads egal ist, bist du herzlich eingeladen, statt continuous_modules lieber Schleifenmodule zu programmieren, in den allermeisten Fällen (tatsächlich allen, die uns bisher eingefallen sind) sind sparsame continuous_modules aber absolut ausreichend und damit eindeutig die bessere Wahl. Achtung: Solltest du tatsächlich einmal in die Verlegenheit kommen, ein Schleifenmodul zu programmieren, solltest du unbedingt möglichst früh in deiner `handle()`-Funktion die Zeile

```
tiane.end_conversation()
```

einfügen, da es sonst passieren kann, dass TIANE eine Konversation, also die Kontrolle über die Sprachein- und ausgabe, für dein Modul reserviert, sodass in diesem Raum kein anderes Modul mehr etwas sagen oder zuhören kann.

Damit continuous_modules von TIANE korrekt erkannt und ausgeführt werden können, müssen sie natürlich wieder in einer bestimmten Form geschrieben werden. Diese besteht diesmal aus zwei Funktionen, die `start()`, `run()` und `stop()` heißen, wobei `start()` und `stop()` aber optional sind: Die Funktion `start()` deines Moduls wird von TIANE nur einmalig beim Laden der Module aufgerufen und dient dazu, von deinem Modul später im Ablauf benötigte Objekte zu initialisieren, damit das später nicht bei jedem Durchlauf aufs neue geschehen muss. Die Funktion erhält als Parameter `tiane` und `local_storage` (die genau dasselbe bedeuten wie schon bei der `handle()`-Funktion von normalen Modulen, bis auf einige Einschränkungen bei `tiane`, dazu später mehr) und muss die Objekte, die sie vorbereitet, auch unbedingt im `local_storage`-Dictionary ablegen und nicht in einer lokalen Variable: Diese werden nämlich von Python „vergessen“, sobald die `start()`-Funktion wieder verlassen wird, stehen also später nicht zur Verfügung. Sollte es übrigens an irgendeiner Stelle darauf ankommen, dass die continuous_modules auf einem TIANE-Gerät in einer bestimmten Reihenfolge gestartet und später auch ausgeführt werden, so kann auch hier der von den normalen Modulen bekannte PRIORITY-Parameter gesetzt werden. Und da man nicht für jedes continuous_module überhaupt Objekte initialisieren muss, ist die `start()`-Funktion rein optional und kann auch einfach kommentarlos weggelassen werden, TIANE wird das Modul trotzdem laden.

Ähnlich verhält es sich mit `stop()`: Diese Funktion erhält die selben Parameter wie `start()` und wird nur einmalig beim beenden des continuous_modules-Threads ausgeführt, also zum Beispiel, wenn das TIANE-Programm auf diesem Gerät angehalten wird oder die Module neu geladen werden sollen, und gibt Modulen die Gelegenheit, externe Ressourcen wie Threads, Objekte oder Dateien sauber zu beenden oder zu schließen, um Probleme zu vermeiden. Auch diese Funktion reagiert auf den PRIORITY-Parameter und ist natürlich auch optional, da sie nur von wenigen Modulen benötigt wird.

Die einzige verpflichtende Funktion, `run()`, ist quasi das Äquivalent zur Funktion `handle()` bei normalen Modulen, enthält also das eigentliche Programm deines Moduls. `run()` erhält auch ähnliche Parameter: `tiane` und `local_storage`, wobei `text` natürlich fehlt, da continuous_modules ja nicht per Sprachbefehl aufgerufen werden, und `tiane` um einige Funktionen beschnitten wurde, dazu gleich mehr. TIANE geht der Reihe nach alle continuous_modules auf einem Gerät durch und führt jeweils deren `run()`-Funktion aus. Da die meisten Module aber gar nicht in so kurzen Zeitabständen ausgeführt werden müssen (wieso sollte sich zum Beispiel ein Kalender zig-mal pro Sekunde aktualisieren?), lässt sich die Ausführung mit dem zusätzlichen Hyperparameter INTERVALL steuern:

```
INTERVALL = 2
```

bedeutet zum Beispiel, dass dein Modul nur maximal alle zwei Sekunden ausgeführt wird, ohne dass andere Module dafür warten müssen. Um Ressourcen auf deiner TIANE-Hardware zu sparen, wird empfohlen, INTERVALL so groß wie möglich zu wählen, wird der Parameter nicht gesetzt, geht TIANE von 0 aus, also sofortiger Ausführung bei jedem Durchlauf. Achtung: INTERVALL gibt nur die minimale Zeit zwischen zwei Ausführungen deines Moduls an, niemals die genaue, da es ja sein kann, dass TIANE gerade noch ein anderes continuous_module ausführt, das eventuell auch noch Zeit benötigt.

Genau das ist auch der Grund, warum die `run()`-Funktion von continuous_modules für die einmalige Ausführung so wenig Zeit wie möglich brauchen sollte: Andere Module mit eventuell betriebswichtigen Aufgaben müssten sonst zu lange warten. Lang laufende Schleifen oder Funktionen oder gar ein Warten auf einen äußeren Zustand gilt es also unbedingt zu vermeiden, solche Vorgänge sollten dringend auf mehrere Durchläufe des Moduls verteilt werden (oder wären der oben angesprochene Spezialfall, der die Verwendung eines „Schleifenmoduls“, das in einem eigenen Thread läuft, rechtfertigen würde). Und um genau solche Wartezeiten zu verhindern, fehlen dem `tiane`-Objekt, das continuous_modules übergeben bekommen, im Vergleich zum `tiane`-Objekt von normalen Modulen auch ein paar Funktionen, nämlich `say()` und `listen()`, die benötigt werden, um TIANE „sprechen“ oder „zuhören“ zu lassen. Diese Funktionen enden nämlich ihrerseits erst wieder, wenn die Sprachausgabe mit dem Aussprechen des Satzes fertig ist bzw. die Spracheingabe ein abgeschlossenes Kommando (+ min. 2 Sekunden Timeout) empfangen und in Text übersetzt hat, was für alle anderen continuous_modules auf dem Gerät eine erhebliche Unterbrechung bedeuten würde.

Wenn dein continuous_module also eine Sprachein- oder -ausgabe braucht, zum Beispiel, weil dein Kalender einen anstehenden Termin ansagen soll, musst du andere Wege gehen. Unter anderem deshalb gibt es die Funktion `tiane.start_module()`, mit der du ein anderes, normales Modul starten kannst, dass dann in einem eigenen Thread läuft und die Ein- oder Ausgabe übernehmen kann, ohne dass die continuous_modules warten müssen. Die Funktion akzeptiert als Parameter unter anderem `room`, `name` und `text` (genaue Dokumentation siehe unten). Du kannst damit ein Modul auf demselben TIANE-Gerät durch Angabe seines Dateinamens als `name` und sogar ein Modul auf einem anderen TIANE-Gerät durch zusätzliche Angabe des Raum- (oder Server-) Namens unter `room` starten und ihm über den Parameter `text` auf Wunsch noch beliebige Anweisungen, wie zum Beispiel den zu sagenden Text, mitteilen. Alternativ kannst du auch nur den `text`-Parameter setzen, damit TIANE mit diesem Text ganz normal die `isValid()`-Funktionen aller Module durchkämmt, bis sich ein Modul für diesen Text zuständig erklärt. Wenn du dich aber für die erstere Variante entscheidest, also für den direkten Aufruf eines eigens dafür entwickelten Moduls, empfiehlt es sich, die `isValid()`-Funktion dieses Moduls immer `False` returnen zu lassen, damit es nicht aus Versehen aufgerufen werden kann. Der direkte Aufruf wird trotzdem funktionieren, da TIANE dafür die `isValid()`-Suche überspringt und einfach direkt die `handle()`-Funktion des angegebenen Moduls aufruft. Der Aufruf eines solchen Ausgabe-Moduls könnte also zum Beispiel so aussehen:

```
tiane.start_module(name='kalender_ausgabe',
                   room='TIANE_SERVER',
                   text='Du hast übrigens gleich einen
                   Termin!')
```

Und das dazugehörige Modul namens kalender_ausgabe auf dem Server so:

```
def isValid(text):
     return False

def handle(text,tiane,local_storage):
     tiane.say(text)
```

Vielleicht merkst du schon, wie nützlich die `start_module()`-Funktion auch für andere Aufgaben ist: Du kannst damit sehr einfach Module im ganzen Haus aufrufen, zum Beispiel um überall das Licht auszuschalten, wenn du das Haus verlässt, und durch die Möglichkeit, Modulen via `text` beliebige Parameter mit auf den Weg zu geben (`text` kann in diesem Fall übrigens auch andere Datentypen enthalten, z.B. Dictionaries), kannst du Module sogar als eine Art globale, im gesamten Netzwerk verfügbare Funktionen betrachten (nur wenn du von diesen Funktionen return-Werte erwartest, müsstest du das anders regeln, zum Beispiel über `local_storage` oder TNetwork, dazu später mehr).

Der local_storage

Nachdem du jetzt die Grundlagen der Modulentwicklung beherrschst, möchtest du vielleicht für deine eigenen Module auch die anderen, in unseren Beispielen nicht näher behandelten Funktionen der TIANE-Modulschnittstelle verstehen. Diese Funktionen werden wir in den nächsten Abschnitten genau dokumentieren, und mit einem ganz speziellen Konstrukt, dem `local_storage`, den die relevante Funktion jedes Moduls als Parameter übergeben bekommt, fangen wir an.

Der `local_storage` ist ein Dictionary, das, wie der Name schon vermuten lässt, für alle Module auf dem lokalen TIANE-Gerät stets synchron ist. Er kann zum Speichern von beliebigen Daten verwendet werden, solange darauf geachtet wird, dass keine von anderen Modulen benötigten Daten überschrieben werden. Der `local_storage` enthält aber bereits zu Anfang Daten über Nutzer, die bei der erstmaligen Einrichtung von TIANE angegeben werden, sowie über Räume und allgemein über dein TIANE-System, und zwar in folgender Struktur:

```
local_storage = {'TIANE_PATH':'/absoluter/Pfad/zum/TIANE/Ordner/
                               des/Geräts',
                 'server_name':'NameDesTianeServers',
                 'system_name':'NameDesSprachassistenten',
                 'home_location':'BeiDerEinrichtungAngegebener
                                  Wohnort'
                 'keys_to_distribute':['users','rooms',
                                       'server_name',
                                       'system_name',...],
                 'activation_phrase': 'BeiDerEinrichtungAnge-
                                       gebenesAktivierungswort',
```

```
'users':{'Ferdi':{
                    'name':'Ferdi',
                    'first_name':'Ferdinand',
                    'last_name':'Krämer',
                    'role':'ADMIN',
                    'uid':1,
                    'room':'Wohnzimmer',
                    'telegram_id': 1234567,
                    'path': '/absoluter/
                             Pfad/zum/
                             Nutzerordner/
                             des/Nutzers
                            (auf dem Server)',
                  },
          'Klara':{
                    'name':'Klara',
                    'first_name':'Klara',
                    'last_name':'Krämer',
                    'role':'AUX_ADMIN',
                    'uid':2,
                    'room':'Schlafzimmer',
                    'telegram_id': 7654321,
                    'path': '/absoluter/
                             Pfad/zum/
                             Nutzerordner/
                             des/Nutzers
                            (auf dem Server)',
                  }},
'rooms':{'Wohnzimmer':{
                        'name':'Wohnzimmer',
                        'users':['Ferdi']},
         'Schlafzimmer':{
                        'name':'Schlafzimmer',
                        'users':['Klara']}},
'TIANE_telegram_name_to_id_table':{
                        'Ferdi': 1234567,
                        'Klara': 7654321},
...}
```

Die persönlichen Daten über die Nutzer können auch mehr oder weniger Felder besitzen, je nachdem, welche Daten bei der Einrichtung angegeben wurden. ‚`role`‘ gibt die Berechtigung dieses Nutzers innerhalb des TIANE-Netzwerks an (zur Zeit noch weitgehend bedeutungslos), ‚`uid`‘ ist eine Zahl, die automatisch angelegt und einfach für jeden Nutzer hochgezählt wird. Die Einträge in ‚`users`‘ im `local_storage` werden beim Start von TIANE automatisch aus den bei der Einrichtung angegebenen Daten zu den einzelnen Nutzern im users-Ordner generiert. Wichtig ist, dass auch sämtliche Daten, die während des Betriebs zu einem Nutzer anfallen, wie z.B. Kalendereinträge etc. auch unter dem entsprechenden Nutzer in der ‚`users`‘-Abteilung im `local_storage` abgespeichert werden, da das die Organisation und Übersicht später stark vereinfacht.

Die Elemente unter dem Schlüssel ‚rooms' im `local_storage` werden komplett automatisch erzeugt, wann immer sich ein Raum mit dem Server verbindet und ihm seine Daten mitteilt, auch hier gehört es aber zum guten Stil, eventuelle zusätzliche Daten über diesen Raum an dieser Stelle zentral zu speichern. Sehr interessant ist die Liste ‚users' für jeden Raum oder das entsprechende Gegenstück ‚room' für jeden Nutzer: In diesen Keys ist die Information gespeichert, wo sich die TIANE-Nutzer eines Haushalts gerade aufhalten, um ihnen zum Beispiel ihre personalisierten user_modules genau dort zur Verfügung zu stellen (sprich, die persönlichen Mails nur dem Empfänger vorzulesen, und nicht „auf gut Glück" irgendwo im Haus). Die Zuweisung der Nutzer zu den Räumen und umgekehrt obliegt allein speziellen Modulen auf dem Server: TIANE bringt dafür von Haus aus das Modul `assign_users.py` mit, das Informationen aus Gesichts- und Stimmerkennung der Räume kombiniert, du bist aber herzlich eingeladen, dieses Modul durch ein eigenes zu ersetzen oder einfach ganz wegzulassen (womit du aber auch auf einen der größten Vorteile von TIANE gegenüber anderen Sprachassistenten verzichten würdest).

Die beiden Schlüssel ‚users' und ‚rooms' im `local_storage`-Dictionary enthalten also die wichtigsten Informationen, die viele Module im gesamten TIANE-System immer wieder aktuell benötigen. Und genau deshalb gilt für diese beiden Schlüssel auch eine Ausnahmeregelung, was die „Lokalität" des Speicherbereichs angeht. Im `local_storage` des Servers gibt es einen zusätzlichen speziellen key, ‚keys_to_distribute', der eine Liste von anderen keys enthält, standardmäßig unter anderem ‚users' und ‚rooms'. Und alle keys (und natürlich die dazugehörigen Daten), die in dieser Liste stehen, werden bei jeder Änderung sofort über das Netzwerk an alle Räume verteilt, damit die Informationen auch dort zur Verfügung stehen! Solltest du diese Funktionalität für eigene Module benötigen, kannst du der Liste beliebig weitere keys hinzufügen.

Ein paar Dinge gibt es bei diesem spannenden System aber unbedingt zu beachten:

Erstens: Die Daten werden nur vom Server an die Räume übertragen. Wenn im `local_storage` eines Raumes unter einem der ‚keys_to_distribute' eine Änderung vorgenommen wird, geht diese unweigerlich bei der nächsten Aktualisierung durch den Server verloren, die Schlüssel ‚users' und ‚rooms' und alle anderen in ‚keys_to_distribute' dürfen also nur von Modulen verändert werden, die auf dem Server laufen, um Datenverlust zu vermeiden.
Zweitens: Daten in solchen vom Server verteilten Schlüsseln dürfen auf keinen Fall Python-Objekte, -Funktionen oder ähnliches enthalten; da diese nicht verteilt werden können, kann es passieren, dass das gesamte TIANE-Server-Programm einfriert!
Solange du aber diese einfachen Regeln beachtest, kannst du auch den `local_storage` benutzen, um sehr schnell und einfach beliebige Daten vom TIANE-Server an alle angeschlossenen Räume zu verteilen.

Allgemein zum `local_storage` sollte außerdem noch gesagt werden, dass Namenskollisionen unbedingt zu vermeiden sind: Wenn verschiedene Module dieselben Schlüsselnamen für ihre Daten im `local_storage` verwenden, überschreiben sie diese immer wieder gegenseitig und werden so höchstwahrscheinlich Probleme bekommen! Es empfiehlt sich daher, für eigene Einträge im `local_storage` möglichst lange, individuelle Schlüssel zu verwenden, um solchen Fällen vorzubeugen. Außerdem sollten möglichst keine Schlüssel verwendet werden, die mit ‚TIANE_' anfangen, dieser Namensraum wird von einigen unserer internen TIANE-Komponenten verwendet. Das Gute daran: solange du

solche Namen in eigenen Modulen nicht verwendest, kannst du sicher sein, dass sie nicht mit unseren mitgelieferten Modulen kollidieren.

Übersicht über alle Attribute und Funktionen der tiane-Klasse

Zum Schluss folgt noch die vollständige Dokumentation aller Attribute und Funktionen des `tiane`-Objekts, das jedes Modul übergeben bekommt, wobei wir auf einige interessante Aspekte natürlich wieder näher eingehen werden. Anmerkung: Im Folgenden werden wir per Sprachkommando oder `start_module()`-Aufruf startbare Module „normale“ Module nennen, continuous_modules manchmal auch „fortlaufende“ Module.

Attribute:

- `tiane.`**`analysis`**
  Typ: Dictionary
  Nur in normalen Modulen verfügbar. Das `analysis`-Dictionary enthält eine vollständige Analyse des gegebenen Kommandos, sofern das Modul per Sprachkommando aufgerufen wurde (und nicht mit `start_module()`). Diese Analyse wird von der Funktion `analyze()` der Klasse `Analyzer` in der Datei „analyze.py“ im TIANE-Ordner des Gerätes angefertigt, die gerne durch eigene Satzanalysetools ersetzt oder besser ergänzt werden kann (eine echte grammatische Analyse wäre zum Beispiel noch hilfreich). Das `analysis`-Dictionary kann zurzeit folgende Informationen über den Satz enthalten:

```
analysis = {'room':'Wohnzimmer',
            'town':'Koblenz',
            'time':{'year':'2018',
                    'month':'01',
                    'day':'01',
                    'hour':'14',
                    'minute':'14'}}
```

  ‚`room`‘ gibt an, ob der Name eines Raumes in deinem TIANE-Netzwerk erwähnt wurde (z.B in „Mach das Licht im Wohnzimmer an“), ‚`town`‘ einen erwähnten Ort (z.B. in „Wie ist das Wetter in Koblenz?“), und ‚`time`‘ eine komplette, absolute Zeitangabe. Gerade dieses `time`-Dictionary ist dabei für eine Vielzahl von Anwendungen besonders wertvoll: `analyze()` rechnet nämlich auch Angaben wie „Übernächste Woche Mittwoch“, „Am 16.10. in zwei Jahren“, „Morgen um drei“ etc. korrekt in das Schema von Jahr, Monat, Tag, Stunde und Minute um. Felder im `analysis`-Dictionary, zu denen die Analyse des Kommandos keinen Inhalt geliefert hat, werden mit ‚`None`‘ gefüllt.

- `tiane.`**`text`**
  Typ: String
  Nur in normalen Modulen verfügbar. Enthält den Text des originalen Kommandos, mit dem das Modul aufgerufen wurde, bzw. den Inhalt, der dem Modul beim Aufruf per `start_module()` unter `text` mitgegeben wurde. Achtung: Wurde das Modul per `start_module()` aufgerufen, ohne den `text`-Parameter zu setzen, enthält dieses Attribut zufälliges Kauderwelsch.

- `tiane.`**`user`**
  Typ: String
  Nur in normalen Modulen sowie in continuous_user_modules verfügbar. Enthält den Namen des Nutzers, der das Modul aufgerufen hat, den Namen des Nutzers, der das Modul aufgerufen hat, welches das Modul aufgerufen hat oder, im Fall von continuous_user_modules, des Nutzers, dem das Modul gehört. Beim Aufruf eines Moduls per `start_module()` kann dem aufgerufenen Modul auch ein beliebiger `user` als Parameter mitgegeben werden. Wenn sich auf all diese Arten kein Nutzer ermitteln lässt (z.B. bei anderen continuous_modules oder wenn das Modul von einem solchen ohne Angabe eines Nutzers aufgerufen wurde) enthält dieses Attribut `None`.

- `tiane.`**`local_storage`**
  Typ: Dictionary
  Enthält den `local_storage` (siehe oben). Eigentlich überflüssig, weil redundant (Module bekommen den `local_storage` ja ohnehin als Parameter übergeben), wurde aber aus Kompatibilitätsgründen mit einigen älteren Modulen behalten.

- `tiane.`**`path`**
  Typ: String
  Enthält den absoluten Pfad zum TIANE-Ordner dieses Geräts.

- `tiane.`**`server_name`**
  Typ: String
  Enthält den Namen deines TIANE-Servers (den Namen, den du bei der Einrichtung von TIANE festgelegt hast, nicht etwa den PC-Namen oder ähnliches).

- `tiane.`**`system_name`**
  Typ: String
  Enthält den Namen deines Sprachassistenten, den du bei der Einrichtung festgelegt hast (z.B. „TIANE“).

- `tiane.`**`room_name`**
  Typ: String
  Enthält in gleicher Manier den Namen des TIANE-Raumes, in dem dieses Modul läuft (nur für Module in Räumen verfügbar).

- `tiane.`**`room_list`**
  Typ: Array
  Enthält eine Liste der room_names (als Strings, s.o.) aller Räume in deinem TIANE-Netzwerk.

- `tiane.`**`userlist`**
  Typ: Array
  Enthält eine Liste der Nutzernamen (Keys im `local_storage[‚users']`-Dictionary) aller Nutzer in deinem TIANE-Netzwerk (als Strings).

- `tiane.`**`users`**
  Typ: Array
  Nur für Module in Räumen verfügbar. Enthält eine Liste der Nutzernamen (als Strings) aller Nutzer, die sich gerade in diesem Raum aufhalten (auch zu finden in `local_storage[‚rooms'][‚NameDesRaums'][‚users']`)

- `tiane.`**`intervall_time`**
  Typ: Int
  Nur für continuous_modules verfügbar. Enthält die per Hyperparameter eingestellte Mindestzeit zwischen zwei Aufrufen (in Sekunden) oder 0, wenn der Parameter nicht gesetzt wurde.

- `tiane.`**`last_call`**
  Typ: Int
  Nur für continuous_modules verfügbar. Enthält einen Timestamp (`time.time()`) vom letzten Aufruf dieses Moduls.

- `tiane.`**`counter`**
  Typ: Int
  Nur für continuous_modules verfügbar. Zählt, wie oft dieses continuous_module seit dem letzten Neustart/Reload bereits aufgerufen wurde.

- `tiane.`**`Analyzer`**
  Typ: Objekt
  Enthält eine Instanz der Satzanalyse-Klasse `Analyzer` aus der Datei „analyze.py“ im TIANE-Ordner des Geräts. Ein Aufruf von `Analyzer.analyze(satz)` mit einem beliebigen Satz als String returnt ein vollständiges `analysis`-Dictionary zu diesem Satz (siehe oben). Auf diese Weise kannst du eine solche Analyse auch zu späteren Nutzereingaben erhalten, nicht nur zu dem originalen Kommando, mit dem das Modul aufgerufen wurde (`tiane.analysis` enthält immer nur die Analyse des originalen Kommandos, auch wenn zwischenzeitlich weitere Nutzereingaben gemacht oder Analysen erstellt wurden).

- `tiane.`**`serverconnection`**
  Typ: Objekt
  Nur für Module in Räumen verfügbar. Enthält ein „TNetwork-Verbindungs-Objekt“ - im Grunde genommen eine universelle, sicher verschlüsselte und sehr einfach zu bedienende Netzwerkverbindung zum Server. Beliebige Daten können in Form eines Dictionaries gesendet werden und landen kurz darauf bei dem empfangenden Server in einem Buffer, bis sie einfach unter Angabe

des gewünschten Keys des Dictionaries wieder abgeholt werden (oder auch nicht) und umgekehrt. Dadurch kannst du sehr einfach auch komplexe Kommunikation zwischen Modulen im gesamten TIANE-Netzwerk einsetzen. Die genauen Funktionen sehen folgendermaßen aus:

- `serverconnection.`**`send`**`(Dictionary)`
  Akzeptiert als Argument ein Dictionary mit beliebigem Inhalt (ausgenommen Objekte!) und beliebig vielen Keys. Das gesamte Dictionary wird daraufhin so bald wie möglich verschlüsselt an den Server übertragen und kann dort abgerufen werden. Zu beachten ist hier, dass Namenskollisionen, ähnlich wie beim `local_storage`, dringend zu vermeiden sind, um keine Daten zu überschreiben; Namen, die mit „TIANE_“ beginnen, sollten daher hier ebenfalls vermieden werden.

- `serverconnection.`**`send_buffer`**`(Dictionary)`
  Akzeptiert als Argument ein Dictionary mit beliebig vielen Keys, die aber alle ein Array als Wert haben müssen. Der gesamte Inhalt jedes Arrays wird dann in der richtigen Reihenfolge an ein unter dem selben Key eventuell noch im lokalen Buffer des Netzwerkobjekts vorhandenes Array angehangen (oder, falls noch kein Array unter dem Key vorhanden war, einfach so übernommen). Diese Funktion ist sehr wichtig, weil es sein kann, dass die TNetwork-Verbindung Daten nicht sofort verschickt, sondern noch für kurze Zeit im Buffer behält (nämlich bis diese Seite wieder mit „senden“ an der Reihe ist). Würdest du nun zum Beispiel Batches aus einem Audio-Stream, den du in einer Schleife nach und nach lädst, oder andere Daten, bei denen es wichtig ist, dass sie vollständig und in der richtigen Reihenfolge ankommen, einfach per `serverconnection.send()` verschicken, könnte es passieren, dass der letzte Batch noch nicht gesendet wurde und so durch den nächsten überschrieben wird, sodass diese Daten auf der anderen Seite fehlen. Durch die Verwendung von `send_buffer()` weist du den TNetwork-Prozess an, diesen Fall zu vermeiden, indem er die Daten nicht überschreibt, sondern anhängt und so einfach beim nächsten Senden als größeren Batch verschickt.

- `serverconnection.`**`read`**`(Key)`
  Akzeptiert als Argument einen beliebigen Dict-Key und returnt sämtliche Daten aus dem lokalen Buffer, die die andere Seite zuletzt unter diesem Key gesendet hat, oder None, falls unter diesem Key keine Daten gefunden wurden (z.B. wenn die Gegenseite diese Daten noch nicht gesendet hat).

- `serverconnection.`**`readanddelete`**`(Key)`
  Gleiche Funktion wie `serverconnection.read()`, löscht die ausgegebenen Daten aber zusätzlich noch aus dem lokalen Buffer. Wann immer du Daten nur einmal empfangen musst (unserer Erfahrung nach in den meisten Fällen), ist die Verwendung von

`readanddelete()` dringend der von `read()` vorzuziehen, um den Buffer möglichst sauber zu halten.

- `tiane.`**`rooms`**
  Typ: Dictionary
  Nur für Module auf dem Server verfügbar. Dieses Dictionary enthält für jeden Raumclient, der mit diesem Server verbunden ist, unter dem jeweiligen Raumnamen (String) als Key eine Raum-Objekt-Instanz für diesen Raum. Dieses Raum-Objekt hat zwei interessante Attribute:
  - `raum.`**`Clientconnection`**
    Typ: Objekt
    Enthält ein „TNetwork-Verbindungs-Objekt“ für die Kommunikation mit dem Raum – das Gegenstück zur `tiane.serverconnection` des jeweiligen Raumes, mit exakt den gleichen Funktionen, nur eben hier für jeden Raum einzeln und zu finden unter `tiane.rooms['NameDesRaums'].Clientconnection`.

  - `raum.`**`users`**
    Typ: Array
    Enthält eine Liste der Nutzernamen (als Strings) aller Nutzer, die sich gerade in diesem Raum aufhalten. Analog zu `tiane.users` im jeweiligen Raum und zu `local_storage['rooms']['NameDesRaums']['users']`.

- `tiane.`**`telegram`**
  Typ: Objekt
  Enthält TIANEs „TelegramInterface“-Instanz, sofern Telegram eingerichtet wurde. Wichtig ist hauptsächlich `tiane.telegram.`**`bot`**, also die telepot-Bot-Instanz, die unter anderem die Funktionen zum Senden komplexer Telegram-Nachrichten und zum Herunterladen empfangener Medien bereitstellt. Nur für Module auf dem Server verfügbar.

- `tiane.`**`telegram_call`**
  Typ: Boolean
  True, wenn dieses Modul per Telegram aufgerufen wurde, andernfalls False. Nur für normale Module verfügbar.

- `tiane.`**`telegram_data`**
  Typ: Dictionary
  Nur für normale Module verfügbar. Enthält das vollständige und von TIANEs TelegramInterface bereits leicht erweiterte telepot-Message-Dictionary der Nachricht, mit der das Modul aufgerufen wurde, falls das Modul via Telegram aufgerufen wurde, andernfalls `None`.

- `tiane.`**`core`**
  Typ: Objekt
  Enthält den „Kern“ von TIANE, nämlich die Hauptinstanz der „echten“ `Tiane`-Klasse des Geräts, von wo aus alle anderen Objekte verwaltet werden.

Ermöglicht extrem tiefe Eingriffe ins System von TIANE, aber nur wenn du wirklich weißt, was du tust. Als Referenz wird auf jeden Fall die Lektüre unserer allgemeinen Dokumentation empfohlen.

Funktionen:

- `tiane.`**`say`**`(text, room=None, user=None, output='auto')`
  Nur für normale Module verfügbar. TIANE spricht den angegebenen `text` per Sprachausgabe aus. Wenn unter `room` ein bestimmter Raum (in Form eines korrekten Raumnamens als String) als Ziel angegeben wurde, wird der Text in diesem Raum gesprochen. Wenn stattdessen ein bestimmter `user` (in Form eines korrekten Nutzernamens als String) als Ziel angegeben wurde, wird der Text in dem Raum ausgegeben, in dem sich dieser Nutzer gerade befindet. Achtung: Falls der Nutzer gerade nicht gefunden werden kann (z.B. weil er nicht zu Hause ist), wird TIANE dabei so lange warten, bis sie den Nutzer wieder findet! Wenn sowohl Raum als auch Nutzer angegeben werden, hat der Raum Vorrang vor dem Nutzer; wenn kein Nutzer angegeben wurde, wird der Nutzer, der das Modul aufgerufen hat, als Standard angenommen; ein einfaches `tiane.say(text)` genügt also, um sehr bequem eine Unterhaltung zu programmieren. Wenn kein Nutzer ermittelt werden konnte, der das Modul aufgerufen hat (bei speziellen Aufrufen per `start_module()`, ein Fall, der weiter oben schon beschrieben wurde), wird stattdessen `room` standardmäßig auf den Namen des Raumes gesetzt, von dem aus das Modul aufgerufen wurde. Wenn das Modul aber zusätzlich auf dem Server liegt, also auch kein Raum ermittelt werden kann, wird sich diese Funktion sofort wieder beenden (und eine kleine Fehlermeldung in der Konsole auf dem Server ausgeben). Ansonsten endet die Funktion, wenn der Text fertig ausgesprochen wurde, damit sich Konversationen noch leichter programmieren lassen. Der zusätzliche Parameter `output` dient zur Steuerung der (Telegram-)Ausgabe: Durch setzen von `output` auf ‚`speech`' oder ‚`telegram`' lässt sich festlegen, dass dieser Text dem Nutzer auf jeden Fall per Sprachausgabe bzw. per Telegram übermittelt werden soll. Die Voreinstellung ‚`auto`' bedeutet, dass TIANE selbst entscheidet, sprich, wenn das Modul per Telegram aufgerufen wurde, wird sie per Telegram antworten, andernfalls per Sprachausgabe. Außerdem kann für diesen Parameter auch ‚`telegram_speech`' angegeben werden, damit TIANE den Text als Telegram-Sprachnachricht verschickt.

- `tiane.`**`listen`**`(user=None, input='auto')`
  Nur für normale Module verfügbar. TIANE hört einem Nutzer zu, nachdem sie ihm einen kurzen Signalton ausgegeben hat, und übersetzt seine Eingabe mit ihrer Spracherkennung in einen String, den diese Funktion returnt. Wenn kein Nutzer angegeben wurde, wird der Nutzer, der das Modul aufgerufen hat, als Standard angenommen, ein einfaches `tiane.listen()` genügt also, um sehr bequem eine Unterhaltung zu programmieren. Sollte sich aber aus speziellen Gründen kein Nutzer ermitteln lassen (siehe oben), returnt diese Funktion „`TIMEOUT_OR_INVALID`". Wenn die Spracherkennung den Satz -

aus welchen Gründen auch immer - nicht erkennen konnte, sich der Nutzer mit der Antwort länger als drei Sekunden Zeit gelassen hat oder irgendein anderer Fehler auftritt, returnt diese Funktion ebenfalls „TIMEOUT_OR_INVALID". Der zusätzliche Parameter `input` dient zur Steuerung der (Telegram-)Eingabe: Durch setzen von `input` auf ‚`speech`' oder ‚`telegram`' lässt sich festlegen, dass die Antwort des Nutzers auf jeden Fall per Spracheingabe bzw. per Telegram erwartet werden soll. Die Voreinstellung ‚`auto`' bedeutet, dass TIANE selbst entscheidet, sprich, wenn das Modul per Telegram aufgerufen wurde, wird sie eine Eingabe per Telegram erwarten, andernfalls per Spracheingabe. Bitte beachte, dass es bei der Eingabe per Telegram KEIN Timeout gibt, sprich, die Funktion wartet ggf. ewig auf eine Antwort des Nutzers. Der Return-Wert „TIMEOUT_OR_INVALID" kann trotzdem auftreten, wenn z.B. ein Medium gesendet wurde, das keinen Text enthält.

- `tiane.`**`telegram_listen`**`(user=None)`
  Gleiche Funktion wie `tiane.listen()`, nur dass hier automatisch Telegram als Quelle angenommen wird und die Funktion nicht nur den Text, sondern das gesamte telepot-Message-Dictionary der Antwort returnt. Nur in normalen Modulen verfügbar.

- `tiane.`**`asynchronous_say`**`(text, room=None, user=None, output='auto')`
  Gleiche Funktion wie `tiane.say()`, nur dass diese Funktion sofort returnt und nicht darauf wartet, dass der Text fertig ausgesprochen wurde. Kann nützlich sein, um Wartezeiten zu überbrücken, während dein Modul etwas berechnet oder im Internet sucht. Nur in normalen Modulen verfügbar.

- `tiane.`**`end_Conversation`**`()`
  Nur in normalen Modulen verfügbar. Diese Funktion sollte dringend aufgerufen werden, wenn ein Modul nicht mehr (durch Sprechen oder Zuhören) mit dem Nutzer interagieren will, aber zum Beispiel für Berechnungen im Hintergrund noch weiterläuft. Nach einem Aufruf von `end_Conversation()` können später trotzdem noch Aufrufe von `listen()` oder `say()` folgen, diese sind aber eventuell mit einer Wartezeit verbunden, da dafür erst eine neue Konversation für dieses Modul reserviert werden muss (und es sein kann, dass die Konversationsrechte mit diesem Nutzer / in diesem Raum zur Zeit noch bei einem anderen Modul liegen).

- `tiane.`**`start_module`**`(user=None, name=None, text=None, room=None)`
  Mithilfe dieser Funktion kannst du aus Modulen heraus andere Module aufrufen. Akzeptiert als Argumente mindestens eine Adresse bestehend aus dem Namen des aufzurufenden Moduls und dem Raum, in dem es liegt (falls nicht in diesem Raum) oder einen Text, mit dem TIANE ganz normal die `isValid()`-Funktionen aller Module durchgeht, um ein passendes zu finden. Sind weder Adresse noch `text` gegeben (oder zumindest nicht korrekt / nicht zutreffend), tut diese Funktion einfach nichts. Wenn aber eine Adresse

angegeben ist, kann unter `text` ein beliebiges Argument angegeben werden (z.B. auch ein ganzes Dictionary), das das adressierte und aufgerufene Modul dann unter `text` wieder abrufen kann. Damit ist es möglich, Module als eine Art globale, im ganzen Netzwerk verfügbare Funktionen zu betrachten. Der zusätzliche Parameter `user` legt fest, welcher Nutzer dem aufgerufenen Modul als „Urheber“ vorgegaukelt werden soll, standardmäßig ist das der Nutzer, der das aufrufende Modul aufgerufen hat (es sei denn, auch dieser lässt sich schon nicht ermitteln, siehe oben). Achtung: Diese Funktion ist recht schnell beendet und zeigt auch sonst durch nichts an, ob ein Modul zum Aufrufen ermittelt werden konnte!

- `tiane.`**`start_module_and_confirm`**`(user=None, name=None, text=None, room=None)`
  Gleiche Funktion wie `start_module()`, nur dass diese Funktion `True` returnt, wenn ein Modul zum Aufrufen ermittelt werden konnte, und `False`, wenn nicht.

Sonstige Modifikationsmöglichkeiten (außerhalb von Modulen)

Neben Modulen bietet der Aufbau von TIANE noch andere Möglichkeiten, mit überschaubarem Aufwand elementare Bestandteile nach deinen Wünschen anzupassen. Zu erwähnen sind hier vor allem die Spracherkennung (Speech-to-text, zu finden in der Datei „stt.py“ im TIANE-Ordner eines Raumes), die Sprachausgabe (Text-to-speech, zu finden in der Datei „tts.py“ im TIANE-Ordner eines Raumes) sowie die Satzanalyse (zu finden in der Datei „analyze.py“ im TIANE-Ordner jedes TIANE-Gerätes). Wenn du diese Dinge gerne ändern möchtest (z.B. um TIANE eine andere Stimme zu verpassen), schau einfach mal in die entsprechenden Dateien rein, dort sind die Details zur Schnittstelle, die deine Implementierung einhalten muss, schnell ersichtlich. Und wenn du etwas Schönes entwickelt hast, ob hier, an einem Modul oder sogar im Kern von TIANE, dann teil es gerne mit uns und anderen über GitHub unter https://github.com/FerdiKr/TIANE/.

Bei Problemen mit der Modulentwicklung oder sonstigen Fragen zum Projekt wende dich außerdem gerne direkt an uns unter jufo.teamkraemer@gmail.com!