

Politechnika Warszawska

W Y D Z I A Ł E L E K T R O N I K I
I T E C H N I K I N F O R M A C Y J N Y C H



Instytut Instytut telekomunikacji

Praca dyplomowa magisterska

na kierunku Telekomunikacja
w specjalności Teleinformatyka i Zarządzanie w Telekomunikacji

Sworzenie platformy do modelowania i uruchamiania zamkniętych pętli
sterowania w Kubernetes

Andrzej Gawor

Numer albumu 300528

promotor
dr inż. Dariusz Bursztynowski

WARSZAWA 2025

Sworzenie platformy do modelowania i uruchamiania zamkniętych pętli sterowania w Kubernetes

Streszczenie. Projekt opisany w niniejszej pracy skupia się na zaproponowaniu oraz przedstawieniu implementacji architektury platformy, która pozwala na modelowanie oraz uruchamianie zamkniętych pętli sterowania w Kubernetes. Genezą projektu jest praca jednego z komitetów ETSI o nazwie "ENI - Experiential Networked Intelligence", która skupia się na ułatwieniu pracy operatora sieci telekomunikacyjnych wykorzystując mechanizmy sztucznej inteligencji w zamkniętych pętlach sterowania. ENI w jednym ze swoich dokumentów dokonuje przeglądu zamkniętych pętli sterowania znanych ludzkości z innych dziedzin.

Naturalnym kolejnym krokiem jest zapropowadzenie platformy, na której operator mógłby takie pętle projektować oraz uruchamiać. W tym celu zdefiniowanie zestaw wymagań oraz założeń dla takiego systemu. Jako środowisko uruchomieniowe wybrano Kubernetes z racji, że jest to system dobrze znany w społeczności oraz sam natywnie używa zamkniętych pętli sterowania. Następnie przeprowadzono obszerną analizę jak za pomocą mechanizmów rozszerzania Kubernetes takich jak "Custom Resources" oraz "Operator" pattern można stworzyć framework umożliwiający modelowanie zamkniętych pętli sterowania. Praca opisuje powstałą platformę, jej architekturę, semantykę składni w definiowanych obiektach, zasady działania, integracje z zewnętrznymi systemami oraz instrukcję jej użytkowania. Omówiona została również implementacja platformy, technologie za nią stojące oraz decyzje podjęte podczas jej powstawania. Finalnie przedstawiono również test działania platformy w praktyce wykorzystując do tego emulator systemu 5G jakim jest Open5GS w połączeniu z UERANSIM. Pracę podsumuję lista wniosków oraz potencjalnych dróg rozwoju platformy.

Słowa kluczowe: Zamknięte pętle sterowania, Kubernetes, Zarządzanie sieciami telekomunikacyjnymi, Automatyzacja, Go, Open5GS, Mikroserwisy

Creation of a platform for designing and running closed control loops in Kubernetes

Abstract. The project described in this thesis focuses on proposing and implementing a reusable architecture (hereafter referred to as "the Platform") that enables the modeling and execution of closed control loops in Kubernetes. The genesis of the project lies in the work of one of the ETSI committees called "ENI - Experiential Networked Intelligence," which aims to simplify the work of telecommunications network operators by leveraging artificial intelligence mechanisms in closed control loops based on metadata-driven and context-aware policies. In one of its documents, ENI reviews closed control loops known to humanity from other fields. A natural next step is to propose a platform on which operators could design and execute such loops.

To achieve this, a set of requirements and assumptions for such a system was defined. Kubernetes was chosen as the runtime environment due to its widespread adoption in the community and its inherent use of closed control loops. An extensive analysis was conducted on how Kubernetes extension mechanisms, such as "Custom Resources" and the "Operator" pattern, could be used to create a framework enabling the modeling of closed control loops.

This thesis describes the developed platform, its architecture, the semantics of syntax in the defined objects, operational principles, integrations with external systems, and a user guide. It also discusses the platform's implementation, the technologies behind it, and the decisions made during its development. Finally, the thesis presents a practical test of the platform's functionality using the Open5GS 5G system emulator in combination with UERANSIM. The work concludes with a list of findings and potential extensions or improvements to the platform.

Keywords: Closed Control Loops, Kubernetes, Managing Telco-Networks, Automation, Go, Open5GS, Microservices

Spis treści

1. Wstęp	7
1.1. Przedmowa	7
1.2. Cel i zakres pracy	7
1.3. Struktura dokumentu	8
2. Stan wiedzy	8
2.1. Wstęp	8
2.2. Historia	8
2.3. ENI	10
2.4. Przegląd pętli sterowania	11
2.4.1. OODA	12
2.4.2. MAPE-K	12
2.4.3. FOCALE	13
2.4.4. COMPA	14
2.5. Teoria Sterowania	15
2.5.1. Układ automatycznej regulacji	15
3. Architektura	16
3.1. Wstęp	16
3.2. Wymagania i założenia	16
3.3. Pojęcia i zasady	17
3.3.1. Wstęp	17
3.3.2. Problem zarządzania	18
3.3.3. System kontroli	18
3.3.4. Pętla sterowania	18
3.3.5. Zamknięta Pętla Sterowania	19
3.3.6. Agenci Translacyjni	19
3.3.7. Workflow pętli	20
3.3.8. Dane i Akcje	21
3.3.9. Open Policy Agent	21
3.4. Instrukcja Użycia	22
3.4.1. Instalacja Luples	22
3.4.2. Implementacja Agentów Translacyjnych	22
3.4.3. Planowanie Workflow Pętli	23
3.4.4. Przygotowanie Elementów Zewnętrznych	23
3.4.5. Wyrażenie workflow pętli	23
3.4.6. Aplikacja plików manifestacyjnych	23
4. Implementacja	24
4.1. Wstęp	24
4.2. Mechanizmy Kubernetes stojące za Luples	24
4.2.1. Kontroler	24
4.2.2. Zasoby niestandardowe	25

4.2.3. Operatory	25
4.2.4. Kubebuilder	26
Bibliografia	27
Wykaz symboli i skrótów	28
Spis rysunków	28
Spis tabel	28
Spis załączników	28

1. Wstęp

1.1. Przedmowa

Wraz z rozwojem telekomunikacji stopień jej skomplikowania jak i mnogość podłączonych urządzeń stale rośnie. Sieci 5G zwiastują obsługę miliardów urządzeń, co sprawia, że tradycjne podejście do zarządzania sieciami staje się niewystarczające. W pewnym momencie manualne operowanie sieciami (ang. *human-driven networks*) stanie się wręcz niemożliwe. Dlatego obserwujemy obecnie zwrot w stronę wirtualizacji oraz automatyzacji sieci. Jednocześnie dynamiczny rozwój sztucznej inteligencji otwiera nowe możliwości. Te dwa czynniki stanowią wspólnie świetny fundament do tego, aby branża sieci telekomunikacyjnych postawiła sobie za cel budowę "inteligentnych" sieci - takich, które są w pełni autonomiczne, samowystarczalne oraz niewymagają nadzoru ludzkiego.

W tym celu ETSI (European Telecommunications Standards Institute) powołało komitet o nazwie ENI - Experiential Networked Intelligence, który ma na celu wypracowanie specyfikacji dla Kognitywnych Systemów Zarządzania Siecią (Cognitive Network Management system). Kognitywny system oznacza taki, który jest w stanie uczyć się i podejmować decyzje bazujące na zebranej wiedzy w sposób przypominający ludzki umysł. ENI opiera swoją architekturę na zamkniętych pętlach sterowania.

Pętlą sterowania, ENI nazywa mechanizm, który monitoruje wydajność systemu lub procesu poddawanego kontroli w celu osiągnięcia pożądanego zachowania. Innymi słowy, pętla sterowania reguluje działanie zarządzanego obiektu. Pętle sterowania można podzielić na zamknięte lub otwarte w zależności od tego czy działanie sterujące zależy od sprzężenia zwrotnego z kontrolowanego systemu lub obiektu. Jeśli tak, pętle nazywamy zamkniętą, jeśli nie - otwartą.

W przypadku architektury ENI, zamknięta pętla sterowania służy jako model organizacji pracy (ang. *workflow*) elementów odpowiedzialnych za sztuczną inteligencję. W dokumencie [1], ENI dokonało przeglądu obiecujących architektur zamkniętych pętli sterowania znanych ludzkości. Naturalnym kolejnym krokiem jest zaproponowanie platformy, na której można takowe pętle zamodelować oraz uruchomić.

1.2. Cel i zakres pracy

Celem pracy jest kontynuacja badań ENI, polegająca na zaproponowaniu *architektury* platformy, na której możliwe będzie modelowanie oraz uruchamianie zamkniętych pętli sterowania. Niniejsza praca nie skupia się na aspektach związanych ze sztuczną inteligencją. Platforma ma jedynie służyć do zamodelowania oraz uruchomienia *workflow* pętli, ale sama nie stanowi środowiska wykonawczego dla jej komponentów.

W zakres pracy wchodzi: sformułowanie wymagań na platformę, opracowanie proponowanej architektury wsparte licznymi badaniami podczas jej powstawania, implementacja PoC (ang. *Proof of Concept*) według proponowanej architektury, przeprowadzenie testów platformy oraz analiza jej potencjału w kontekście dalszego rozwoju. Po publikacji praca

może stanowić podstawy do konkretnych implementacji Kognitywnych Systemów Zarządzania Siecią, specyfikowanych przez ENI.

1.3. Struktura dokumentu

Dokument został podzielony na 6 sekcji. Druga sekcja przedstawia bardziej szczegółowo niż we wstępnie badaniu podjęte przez ENI. Stanowi ona nieco wprowadzenie teoretyczne oraz pojęciowe, aby ułatwić przekaz w dalszej części pracy. Sekcja trzecia zawiera opis proponowanej architektury. Sekcja czwarta opisuje implementację PoC platformy oraz napotkane wyzwania podczas formułowania architektury, które celowo zostały zebrane w jedno miejsce i umieszczone oddziennie w celu łatwiejszej lektury. Sekcja numer pięć przedstawia użycie platformy w praktyce przy okazji stanowiąc jej test. Na koniec, w sekcji szóstej przeprowadzono analizę zaproponowanej architektury, przedstawiono jej możliwości oraz ograniczenia oraz wskazano kierunki potencjalnego rozwoju.

2. Stan wiedzy

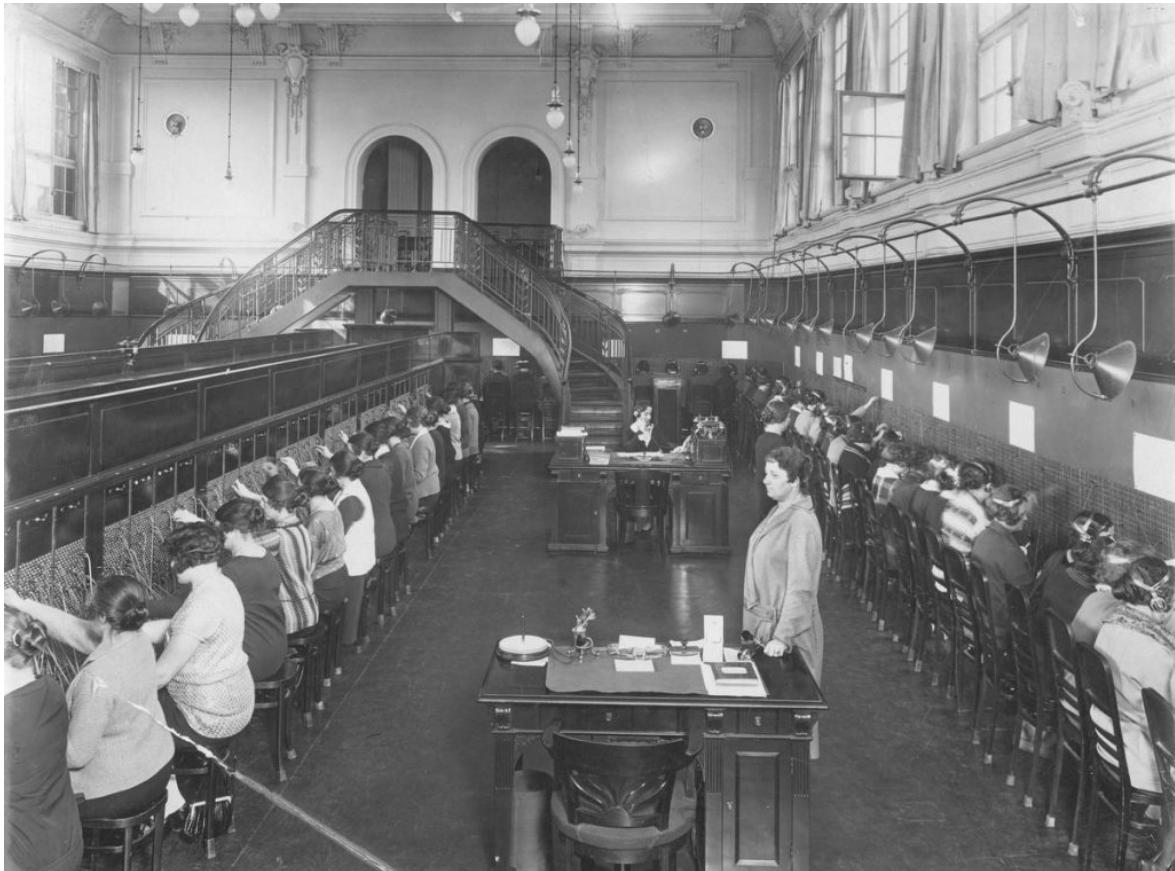
2.1. Wstęp

Sekcja ta stanowi krótki wstęp historyczny problemu jakiego podjęło się ENI, prezentuje ich badania relevantne z punktu widzenia pracy oraz zawiera wstęp z Teorii Sterowania - dziedziny, która zajmuje się zamkniętymi pętlami sterowania.

2.2. Historia

Na początku XX wieku, gdy sieci telekomunikacyjne dopiero się rozwijały, wszystkie połączenia zestawiane były ręcznie. W momencie, gdy abonent podniósł słuchawkę telefonu, jego aparat wysyłał sygnał do lokalnej centrali telefonicznej. Na tablicy świetlnej zapalała się lampka informująca telefonistkę o próbie połączenia. Telefonistka odbierała, pytając, z kim abonent chce się połączyć. Następnie wprowadzała odpowiednią wtyczkę do odpowiedniego gniazda na tablicy rozdzielczej, zestawiając fizyczne połączenie między dwoma liniami. Jeśli rozmowa miała się odbyć na większą odległość (np. między miastami) połączenie przekazywane było przez kolejne centrale. Każda centrala po drodze wymagała ręcznej obsługi przez pracujące w nich telefonistki.

Dziś taki scenariusz wydaje się wręcz absurdalny, a oferty pracy dla telefonistek dawno już zniknęły z tablic ogłoszeń. Praca wykonywana przez telefonistki (czyli komutacja łączy) nadal jest potrzebna do prawidłowego funkcjonowania sieci telekomunikacyjnych, lecz wykonywana jest przez programy komputerowe w sposób w pełni zautomatyzowany. Ręczna komutacja była pierwszym krokiem w kierunku rozwoju globalnych sieci telekomunikacyjnych i choć z dzisiejszej perspektywy wydaje się być bardzo pracochłonna i ograniczająca, bez niej nie powstałyby fundamenty, na których oparto późniejsze systemy automatyczne. Jest to przykład tego, jak technologia stopniowo uwalniała człowieka od bezpośredniej obsługi różnych systemów (dając mu przestrzeń na rózwój w innych obszarach).



Rysunek 2.1. Pracowniczki warszawskiej centrali telefonicznej z końca lat 20. XX wieku

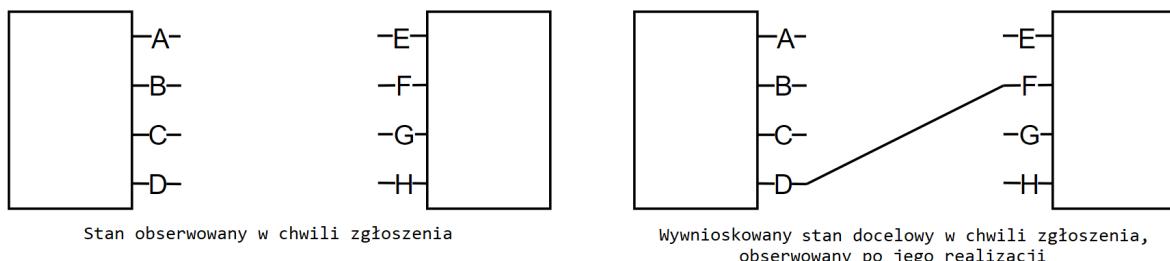
Na zasadzie indukcji możemy przyjąć, iż dziś znajdujemy się w podobnym położeniu - sieci telekomunikacyjne wciąż wymagają bezpośredniego zarządzania przez człowieka. Po prostu granica tego styku - system-człowiek - jest mocno przesunięta. Dziś człowiek spotyka się z systemami dużo bardziej złożonymi, a zarazem dającymi dużo więcej możliwości. Możliwe, że nie istnieje ostateczny punkt styku i systemy telekomunikacyjne zawsze będą wymagały nadzoru ludzkiego. Niekwestionowanie od tej kwestii przesunięcie tej granicy w czasach telefonistek, a dziś wymaga automatyzacji innego rodzaju.

O telefonistkach możemy powiedzieć, że zarządzały one pracą systemu telekomunikacyjnego¹. Regulowały one jego działanie. W momencie zgłoszenia zastawały one system w pewnym stanie i w ramach realizacji zgłoszenia musiały doprowadzić system do nowego (docelowego) stanu. Cała ich praca tak naprawdę polegała na "wnioskowaniu" (ang. *reason*) jak ma wyglądać stan docelowy, a następnie na wykonaniu *akcji*, która doprowadzała system do tego stanu.

Pomówmy trochę o *wnioskowaniu*. Jak ono przebiega? Po pierwsze telefonistka otrzymuje *dane*. Dane są to fakty lub statystki zebrane razem w celu analizy. Telefonistka zbiera dane w formie faktów jakie miały miejsce. Zadzwonił aparat telefoniczny, telefonistka podniosła słuchawkę i usłyszała w niej, że abonent chce się połączyć ze swoim szwagrem mieszkającym w kamienicy pod numerem 35 na ulicy polnej. Dostępne dane, telefonistka

¹ t.j. siecią PSTN

2. Stan wiedzy



Rysunek 2.2. Dwa stany systemu: przed, oraz po realizacji zgłoszenia o treści: "D"chce zadzwonić do "F"

przekała na *informację*. Informacja to dane pozbawione formy przekazu. Równie dobrze, rozmowa z abonentem mogła przebiec zupełnie inaczej lub telefonistka mogła bytrzymać listowną prośbę o realizację połączenia. Zupełnie inne dane dostarczyły by tej samej informacji, czyli to, że abonent spod konkretnego łączka, chce się dodzwonić do abonenta spod innego, konkretnego łączka. Gdy telefonistka otrzyma informacje na temat zgłoszenia, łączy ją z dotychczasowo posiadanymi informacjami, jak rozmieszczenie łączów na tablicy komutacyjnej. Następnie używa swojej *wiedzy*, czyli zestawu wzorców, które pozwalają jej wytlumaczyć oraz przewidzieć co się wydarzyło, dzieje w tej chwili lub może wydarzyć w przyszłości. Wiedza opiera się na informacji oraz umiejętnościach pozyskanych dzięki doświadczeniu lub edukacji. Obserwuje ona aktualny stan systemu i *rozumując*, na podstawie wiedzy wyobraża sobie stan docelowy (pożądany) oraz definiuje *akcje*, jakie należy przeprowadzić na systemie, aby osiągnął on ten stan.

W przypadku komutacji proces wnioskowania jest względnie prosty, dający się do zapisania w języku programowania. Dlatego też wraz z rozwojem technologicznym dokonano przejścia na automatyczne centrale telefoniczne. Jednakże opis wyżej nadal pozostaje ważny. Do centrali trafiają dane w postaci sygnału na jednym z łącz. Sygnał przedstawia numer MSISDN abonenta końcowego. Centrala przekuwa te dane na informacje, a następnie program w jej pamięci dokonuje wnioskowania. Dzięki programowalnym przełącznicom komutacyjnym również wykonanie *akcji* (czyli połączenie ze sobą dwóch łącz) jest możliwe bez udziału człowieka. W dzisiejszych czasach proces wnioskowania wykonywany przez ludzi nadzorujących system telekomunikacyjne już nie jest tak prosty. Dzięki wirtualizacji i programowalności funkcji sieciowych każdy rodzaj *wnioskowania* jaki można skondensować do programu komputerowego został już poddany temu procesowi. Teraz akcje wykonywane przez człowieka rzeczywiście wymagają jego inteligencji². Dlatego więc następnym krokiem w rozwoju telekomunikacji jest stworzenie intelligentnych sieci.

2.3. ENI

ENI rozwija specyfikacje w kierunku Kognitywnego Systemu Zarządzania, który będzie regulował działanie sieci poprzez wykorzystanie technik sztucznej inteligencji, takich jak uczenie maszynowe (ang. *machine learning*) oraz wnioskowanie (ang. *reasoning*).

² lub sprawczości w świecie fizycznym, której komputerom brakuje

Z poprzedniej sekcji wiemy, że aby *wnioskować* potrzebna jest wiedza. Skąd taki system pozyskiwałby wiedzę? Otóż, dzieje się to poprzez proces *machine learning'*u. Pierwsza faza czyli tzw. *trening*, odbywa się jeszcze przed wdrożeniem systemu. Ale system również może uczyć się "z doświadczenia" (ang. *experience*) po wdrożeniu, kiedy już jest w pełni operacyjny. Stąd mówimy o empirycznej inteligencji sieciowej (ang. *experiential networked intelligence*).

Czyli z jednej strony wiedza potrzebna do wnioskowania, a w ostateczności podejmowania decyzji płynęłaby z samej sieci, co w przypadku telefonistki równoważne jest zapalającej się lampce na tablicy świetlnej. Z drugiej strony telefonistka ma również niejako wbudowaną w siebie wiedzę o tym, jak zorganizowane są łącza na przełącznicy komutacyjnej. Podobnie sieć musi mieć pewną części wiedzy nieco narzuconą z góry. Ta część wiedzy stanowi jako zestaw reguł używanych do zarządzania i kontrolowania stanu sieci. Takie reguły nazywamy *politykami*.

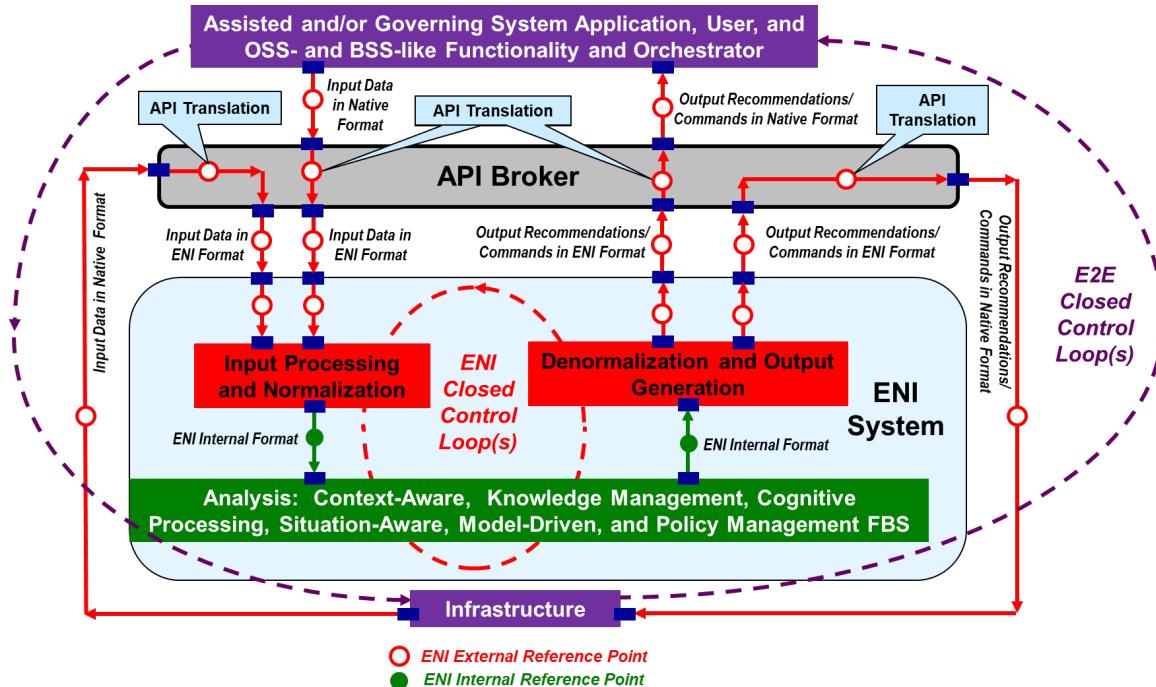
Polityki mogą płynąć od: aplikacji zarządzających siecią, użytkowników sieci, systemów OSS/BSS lub Orkiestratora. Ważne jest to, aby polityki były *świadome kontekstu*. Pozwoli to na stworzenie systemu kognitywnego, czyli takiego, który uczy się, wnioskuje oraz podejmuje decyzje w sposób przypominający ludzki umysł. Taki system z kolei jest w stanie w dużym stopniu odciążyć operatora i zautomatyzować zadania jak:

- dynamiczne przydzielanie zasobów (ang. *dynamic resource allocation*),
- równoważenie obciążenia (ang. *load balancing*),
- zarządzanie wydajnością energetyczną (ang. *energy efficiency management*),
- samo naprawiające się sieci (ang. *self-healing networks*),
- optymalizacja jakości doświadczeń użytkowników (ang. *QoE optimization*),
- egzekwowanie polityk (ang. *policy enforcement*),
- zapewnienie zgodności z regulacjami (ang. *regulatory compliance*)
- i wiele innych.

Wysokopoziomową architekturę funkcjonalną ENI pokazano na rysunku 2.3. Jej działanie można opisać jako dwie *zamknięte pętle sterowania*. Za pomocą zewnętrznej, system zarządzania (użytkownik, OSS/BSS, orkiestrator) reguluje pracę sieci (infrastruktury). Dokonuje tego nie bezpośrednio, lecz za pomocą ENI System, który sam wewnętrznie również posiada zamkniętą pętlę sterowania. Wewnętrzna pętla podczas wnioskowania bierze pod uwagę nie tylko wiedzę płynącą bezpośrednio z sieci, ale również tę od orkiestratora. Wszystkie komponenty odpowiedzialne za sztuczną inteligencję są umieszczone na rysunku 2.3 w zielonym prostokącie i stanowią *workflow* wewnętrznej pętli.

2.4. Przegląd pętli sterowania

Przedstawiona w poprzedniej sekcji architektura jest jedynie funkcjonalna i zawiera w sobie dużo abstrakcji. Ostateczna postać wewnętrznej zamkniętej pętli sterowania w ENI system jest jeszcze w trakcie specyfikacji. ENI w dokumencie [1] dokonało przeglądu istniejących, możliwych do zaaplikowania architektur zamkniętych pętli sterowania. Niniejsza praca dyplomowa chce zaproponować architekturę do modelowania oraz uruchamiania



Rysunek 2.3. Wysokopoziomowa architektura funkcjonalna ENI

pętli w ENI System, dlatego ważne jest, aby poznać jej składowe elementy oraz *workflow* przykładowych pętli. Domyślamy się, że konkretne workflow pętli będzie znane dopiero przy konkretnym wdrożeniu ENI System, a wdrożeniu będą różniły się między sobą.

2.4.1. OODA

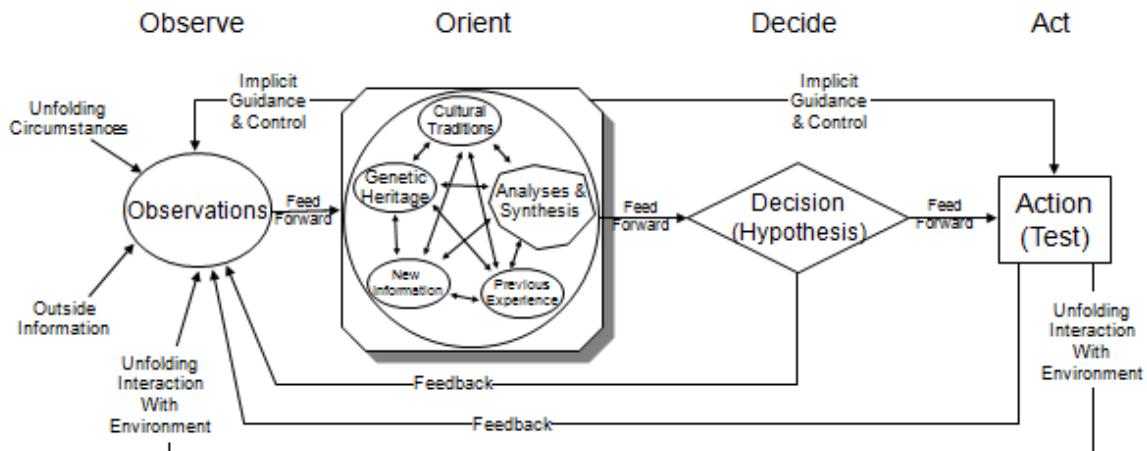
Pętla autorstwa Johna R. Boyda składa się z 4 części: "Observe", "Orient", "Decide", "Act". Stanowi podsumowanie jego myśli na temat strategii, taktyki oraz procesów decyzyjnych. John R. Boyd był pułkownikiem Sił Powietrznych USA i jednym z najbardziej wpływowych myślicieli wojskowych XX wieku. Mimo, iż był wojskowym jego koncepcja składająca się z nieustannej obserwacji, orientacji, podejmowania decyzji i działania jest kluczowa w każdej formie walki. Może być również stosowana w polityce, biznesie, życiu codziennym. Główną koncepcją jest ciągłe dostosowywanie się do zmieniających się warunków oraz podważanie istniejących założeń. Koncepcja ta świetnie nadaje się do adaptacyjnych oraz kognitywnych systemów zarządzania.

Architektura pętli OODA przedstawiona jest na rysunku 2.4

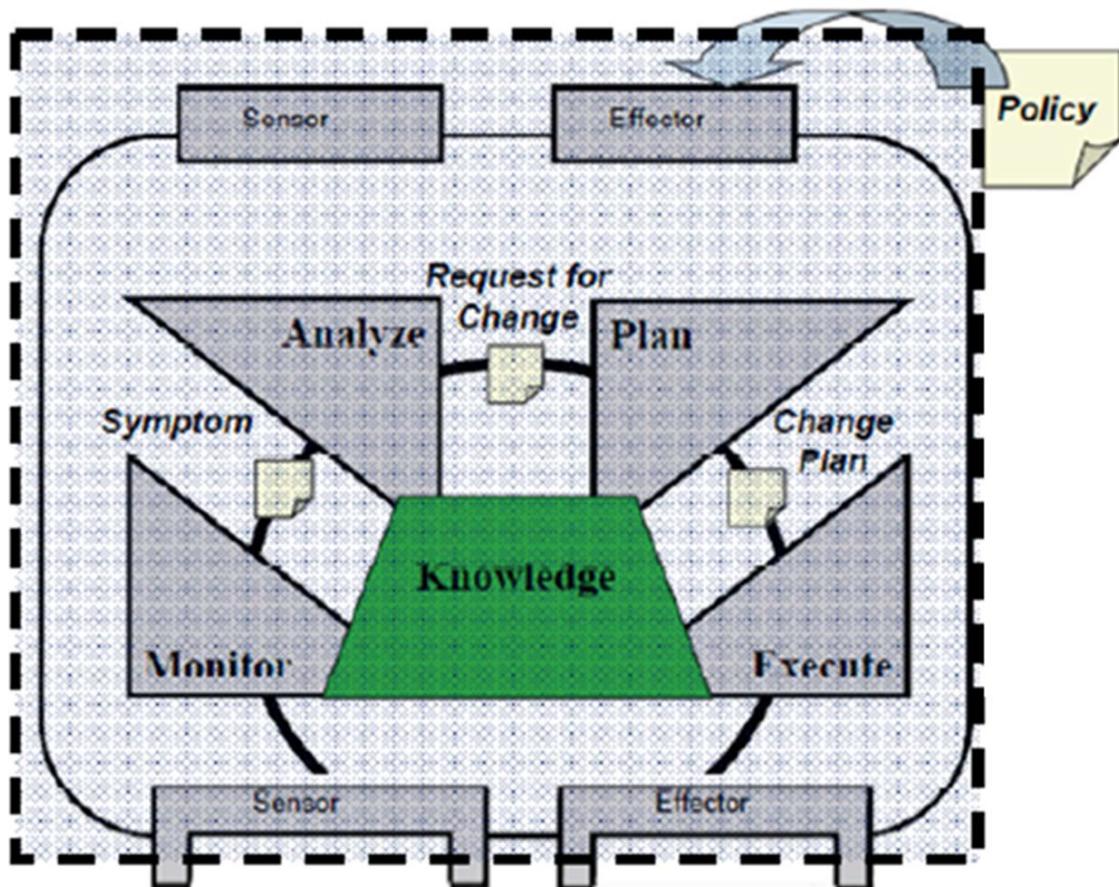
2.4.2. MAPE-K

MAPE-K jest to pętla opracowana przez IBM w dziedzinie autonomicznego przetwarzania i systemów samoadaptujących się. Główne jej etapy to Monitorowanie, Analiza, Planowanie, Wykonanie, a każdy z nich ma dostęp do wspólnej bazy wiedzy.

Architektura pętli MAPE-K przedstawiona jest na rysunku 2.5



Rysunek 2.4. Architektura pętli OODA



Rysunek 2.5. Architektura pętli MAPE-K

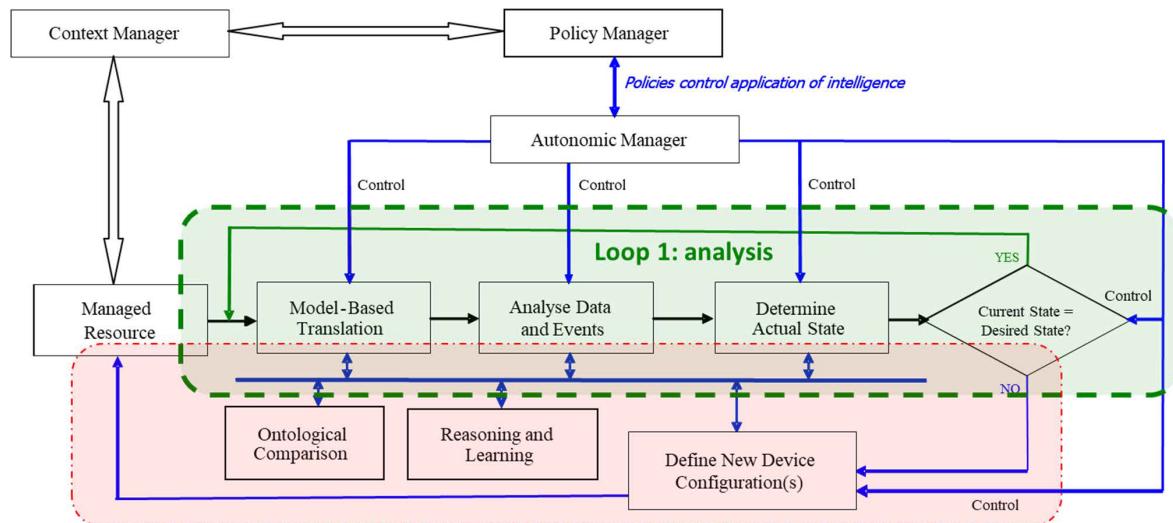
2.4.3. FOCALE

Nazwa tej pętli pochodzi słów: Podstawa (ang. *Foundation*), Obserwacja (ang. *Observation*), Porównanie (ang. *Comparison*), Analiza (ang. *Analysis*), Uczenie (ang. *Learning*), Wnioskowanie (ang. *rEason*). Niespotykanym dotąd elementem może być Podstawa, która jest statyczną częścią modelu i określa strukturę oraz podstawowe zasady systemu. Ta

2. Stan wiedzy

architektura jako pierwsza proponuje też element odpowiedzialny za uczenie maszynowe (ang. *machine learning*).

Architektura pętli FOCALE przedstawiona jest na rysunku 2.6

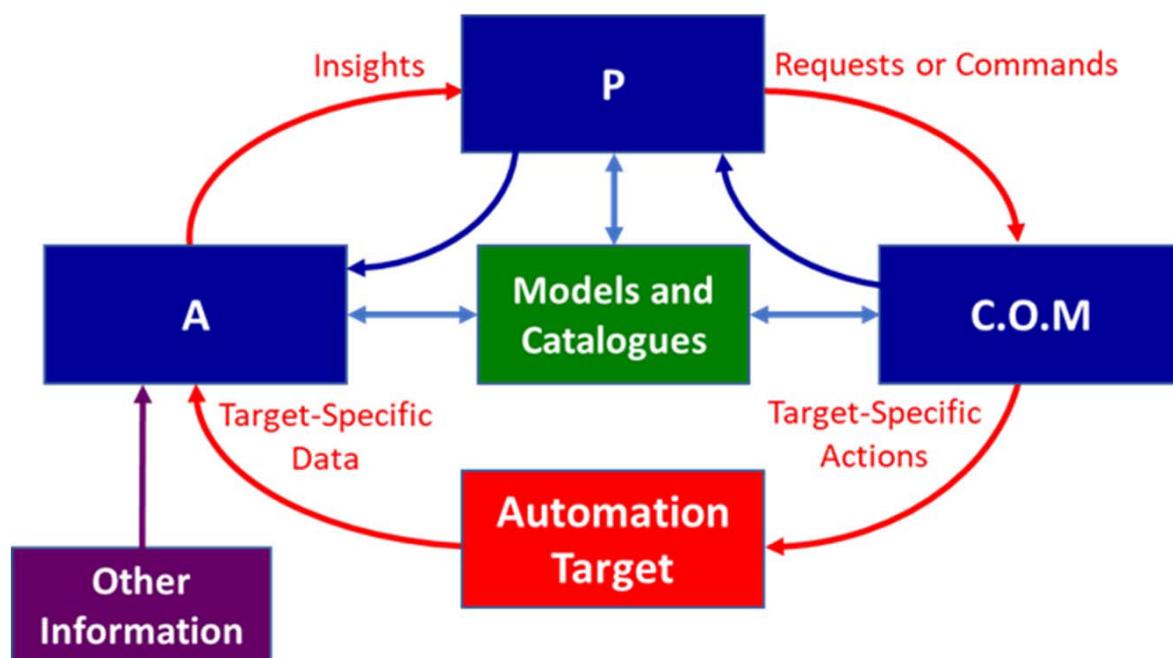


Rysunek 2.6. Architektura pętli FOCALE

2.4.4. COMPA

Elementy tej pętli to: Analityka (ang. *Analytics*), Polityki (ang. *Policies*) oraz Sterowanie, Orkiestracja i Zarządzanie (ang. *Control, Orchestration, Management*)

Architektura pętli COMPA przedstawiona jest na rysunku 2.7



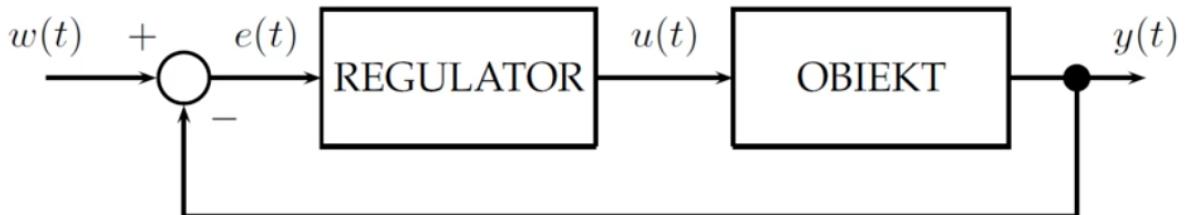
Rysunek 2.7. Architektura pętli COMPA

2.5. Teoria Sterowania

Teoria sterowania (ang. *control theory*) jest to dziedzina inżynierii oraz matematyki stosowanej, która mierzy się z kontrolowaniem układów dynamicznych³ w procesach inżynierijnych i maszynach. Jej celem w danym problemie jest opracowanie modelu lub algorytmu, który zarządza sygnałami wejściowymi do systemu tak, aby osiągnął on stan docelowy.

W tym celu potrzebny jest regulator (ang. *control system*). Regulator zarządza, steruje, kieruje lub reguluje zachowanie innych urządzeń lub systemów, wykorzystując pętle sterowania. Regulator monitoruje kontrolowaną zmienną procesu (ang. *process variable*) (PV) i porównuje ją z wartością odniesienia (ang. *set point*) (SP). Różnica między rzeczywistą a pożądaną wartością zmiennej procesu, nazywana sygnałem błędu lub uchybem regulacji jest stosowana jako sprzężenie zwrotne w celu wygenerowania sygnału sterującego, które doprowadzi kontrolowaną zmienną procesu (PV) do wartości równej punktowi odniesienia (SP).

2.5.1. Układ automatycznej regulacji



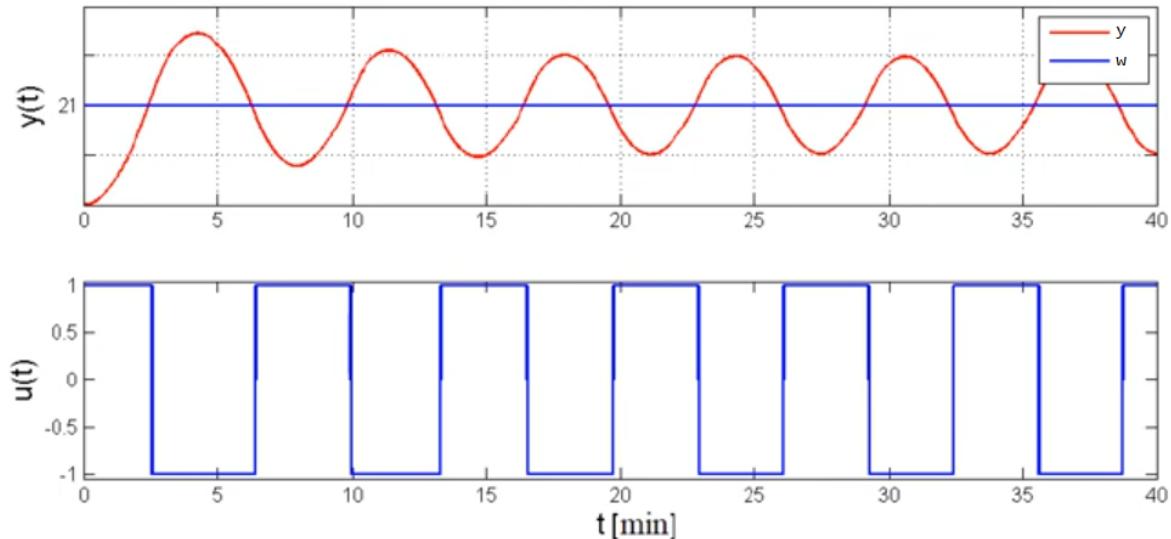
Rysunek 2.8. Układ automatycznej regulacji (UAR)

Jeśli chcemy kontrolować wyjście danego obiektu zamykamy układ sprzężeniem zwrotnym. Wykonujemy pomiaru sygnał wyjściowygo $y(t)$ i porównujemy z sygnałem odniesienia $w(t)$ (ang. *set point*). Otrzymujemy w ten sposób *uchyb regulacji* $e(t)$, który trafia do regulatora. Regulator przetwarza dostarczony mu uchyb i na tej podstawie wytwarza sygnał sterujący *zarządzanym obiektem*, czyli sygnał $u(t)$ zwany również wymuszeniem.

Przykładem może być układ regulacji temperatury obiektu, gdzie regulator ma postać przekaźnika dwupołożeniowego. Jeżeli jest za zimno złącza się stan wysoki, czyli ogrzewanie. Jeżeli jest za gorąco włącza się stan niski czyli chłodzenie. Punktem odniesienia $w(t)$ jest tutaj temperatura 21 stopni. Gdy *pomiar* $y(t)$ znajduje się poniżej tej wartości, obliczany *uchyb regulacji* $e(t)$ przyjmuje wartość dodatnią. Regulator jest zaprogramowany w taki sposób (wykonuje taki algorytm), że w takim przypadku jego wyjście $u(t)$ przyjmuje położenie wysokie (wartość 1). Obiekt z kolei, pobudzony takim sygnałem na wejściu podnosi swoją temperaturę. Analogiczny tok rozumowania można przeprowadzić w przypadku, gdy $e(t)$ jest ujemne.

Oczywiście obiekt nie jest systemem dyskretnym, a cały proces dzieje się w układzie dynamicznym, stąd na wykresie 2.9 pomiaru obserwujemy sinusoidę.

³ https://en.wikipedia.org/wiki/Dynamical_system



Rysunek 2.9. Wykresy pomiaru, punktu odniesienia oraz wymuszenia w czasie

3. Architektura

3.1. Wstęp

Niniejsza sekcja przedstawia zaproponowaną w pracy architekturę ENI System, t.j. platformy, na której jest możliwe modelowanie oraz uruchamianie workflow zamkniętych pętli sterowania. Platformie, nadano nazwę w celu ułatwienia jej opisu. Nazwa brzmi "Lupus". Powstała od przekształcenia angielskiego słowa "loops" oznaczającego pętle, oraz od zakotwiczenia o wyraz mający znaczenie nadające się na "maskotkę" projektu. "Lupus" po łacinie oznacza wilka, a po angielsku "toczeń rumieniowaty", stąd w logo projektu zarumieniony wilk.

Architektura w rozumieniu ENI jest to "zbiór reguł i metod opisujących funkcjonalność, organizację oraz implementację systemu". Niniejsza sekcja pomija aspekt implementacji, która jest omówiona w następnej sekcji.

3.2. Wymagania i założenia

Po pierwsze Lupus ma przyjąć rolę regulatora (ang. control system) znanego z teorii sterowania. Regulowanymi systemami mają być w tym przypadku systemy telekomunikacyjne. Aby odegrać rolę ENI System, platforma musi być w stanie zamodelować oraz uruchomić dowolną zamkniętą pętlę sterowania, zwłaszcza te zawarte w [1].

Z racji ogólności systemu oraz zwiększenia szansy na dobre przyjęcie w społeczności wybrano Kubernetes jako infrastrukturę dla platformy. Kubernetes natywnie używa zamkniętych pętli sterowania w swojej warstwie sterowania (ang. *control plane*), z których użytkownik jest w stanie skorzystać za pomocą mechanizmów rozszerzeń Kubernetes takich jak Custom Resource Definitions (CRD) oraz Operator Pattern. Są to mechanizmy dobrze znane w branży.

Wybór Kubernetes narzuca jednakże pewne ograniczenie. Regulowane procesy nie mogą odbywać się w czasie rzeczywistym. Warstwa sterowania Kubernetes działa nieco wolniej.

Lupus musi być "data-driven" co na polski można przetłumaczyć jako "napędzany danymi". Oznacza to, że platforma nie może narzucać żadnej postaci logiki pętli. Warstwa sterowania Kubernetes musi być w stanie interpretować zamiary użytkownika platformy, który może wyrazić dowolną pętle. Zamiary te wyrażone są właśnie w danych.

Logikę pętli możemy podzielić na dwie części: workflow pętli oraz części obliczeniowa. Workflow jest to zdefiniowane elementów oraz relacji między nimi. Częścią obliczeniową za to nazywamy procesowanie wykonywane przez konkretne elementy. Z racji podejścia "data-driven" nie możemy zaszyć części obliczeniowej w warstwie sterowania Kubernetes. Dlatego elementy odpowiedzialne za części obliczeniową są "na zewnątrz" pętli Lupus, przykładowo są to serwery HTTP, do których Lupus wykonując pętle może się odwoływać na rzecz jej logiki.

Z tego opisu powstaje garść wymagań oraz założeń, które z pewnością nie są zbiorami rozłącznymi i wielu miejscach się zacierają, jednakże warto wyszczególnić je w sposób wylistowany poniżej, aby móc łatwiej się do nich odwoływać w dalszej części pracy:

1. Lupus jest skierowany do branży telekomunikacyjnej.
2. Lupus umożliwia modelowanie i uruchamianie dowolnych architektur zamkniętych pętli sterowania, w szczególności tych zaproponowanych w [1].
3. Lupus zarządza procesami, które nie wymagają regulacji w czasie rzeczywistym (są "non-realtime").
4. Lupus jest zaimplementowany na bazie Kubernetes, wykorzystując jego *Controller Pattern*.
5. Lupus jest oparty na danych (*data-driven*), co oznacza, że nie narzuca i nie ma wbudowanej żadnej domyślnej logiki pętli.
6. Faktyczne komponenty przetwarzające w pętli (część obliczeniowa) Lupus są zewnętrzne względem niego (np. serwery HTTP, szczególnie Open Policy Agent).
7. Lupus powinien być w stanie regulować pracę dowolnego systemu teleinformatycznego bez żadnych jego modyfikacji
8. Zamodelowanie pętli oraz wyrażenie jej **Workflow** w Lupus nie powinno wymagać umiejętności technicznych

Powyższa lista nazwana jest Wymaganiem i referowana w dalszej części dokumentu.

3.3. Pojęcia i zasady

3.3.1. Wstęp

Ta podsekcja dokumentuje Lupus w formie wyjaśniania zdefiniowanych na jego potrzeby pojęć, konceptów i zasad. Są one punktem wyjścia do dokładniejszych specyfikacji i zarazem zrozumienia architektury systemu. Sekcja ta zawiera wiele linków do definicji znajdujących się w załączniku i nie wszystkie pojęcia tłumaczone są w tej sekcji.

3.3.2. Problem zarządzania

W rzeczywistym świecie często napotkamy sytuacje, w których byłoby dobrze, gdyby praca jakiegoś systemu mogła być stale regulowana. Na przykład:

- chcielibyśmy, aby samochody miały funkcję regulującą pracę silnika w celu utrzymania stałej prędkości,
- przydałoby się, gdyby lodówka mogła utrzymywać chłodną, ale nie ujemną temperaturę, niezależnie od tego, jak często otwierane są drzwi lub jaka jest temperatura na zewnątrz w danym dniu,
- byłoby korzystne, gdyby serwer w chmurze mógł zagwarantować, że aplikacja z wystarczającymi zasobami do pokrycia potrzeb użytkowników będzie uruchomiona i działała.

Problemy wymienione powyżej można traktować jako **problemy zarządzania** (ang. *management problem*). Nazwa bierze się stąd, że nie ma żadnych technicznych ograniczeń uniemożliwiających osiągnięcie tych celów. Wszystkie wymienione systemy są odpowiednio wyposażone np. możemy dodać więcej paliwa do silnika lub dostarczyć więcej mocy do sprężarki w lodówce. Problem leży w faktycznym wykonaniu tych czynności w odpowiednich momentach np. dodanie więcej paliwa, gdy auto zwalnia lub dostarczenie większej mocy gdy temperatura w lodówce wzrasta. Dlatego jest to problem czystego zarządzania.

Za to system, którym chcemy zarządzać nazywamy **zarządzanym systemem** (ang. *managed system*).

3.3.3. System kontroli

System kontroli (ang. *Control System*⁴) to system, który reguluje pracę **zarządzanego systemu**. Na przykład:

- tempomat, który reguluje pracę silnika w celu utrzymania stałej prędkości,
- lodówka, która reguluje pracę sprężarki w celu utrzymania stałej, chłodnej temperatury
- Kubernetes, który reguluje liczbę działających Podów, aby utrzymać pożądaną dostępności aplikacji

Innymi słowy mówiąc **System kontroli** rozwiązuje **Problem Zarządzania**.

3.3.4. Pętla sterowania

Ogólną architekturą **Systemów Kontroli** używaną do rozwiązywania **Problemów Zarządzania** jest **Pętla Sterowania**.

Pętle sterowania są klasyfikowane w zależności od tego, czy wykorzystują mechanizmy sprzężenia zwrotnego (ang. *feedback mechanism*):

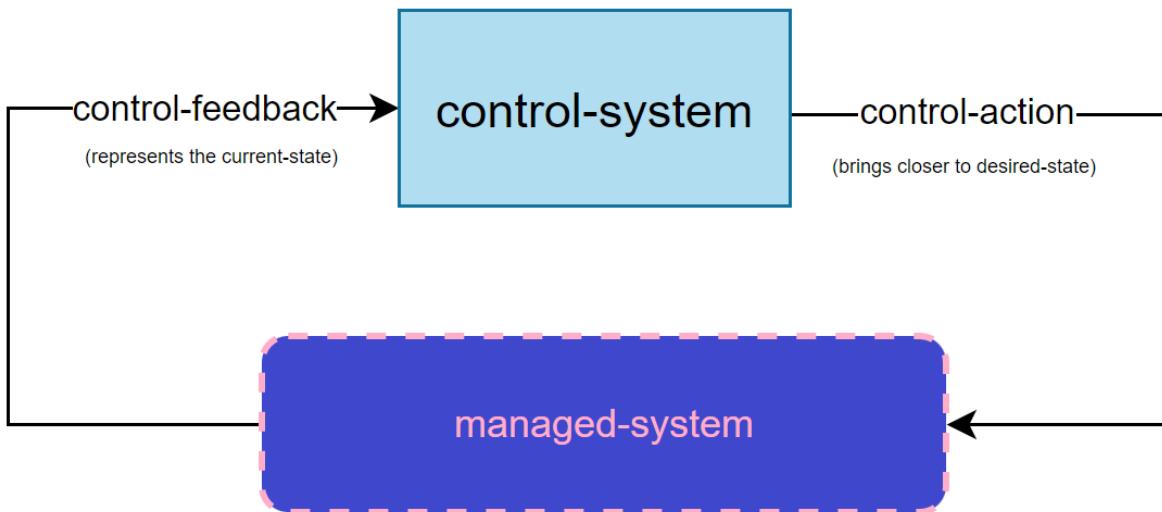
- **Otwarte Pętle Sterowania:** Akcja Sterująca (ang. *Control Action*) (czyli wejście do **Systemu Zarządzanego**) jest niezależne od wyjścia **Systemu Zarządzanego**.
- **Zamknięte Pętle Sterowania:** Wyjście **Systemu Zarządzanego** jest sprężane do wejścia **Systemu Kontroli** i wpływa na Akcję Sterującą.

⁴ innym tłumaczeniem na polski jest "regulator"

W Luples bierzemy pod uwagę wyłącznie zamknięte pętle sterowania.

3.3.5. Zamknięta Pętla Sterowania

Jest to punkt wyjściowy (startowy) dla naszej architektury referencyjnej (rys 3.1).



Rysunek 3.1. Architektura referencyjna dla Luples

Architektura (rys. 3.1) należy czytać mając na uwadze następujące definicje:

- **System Sterowania (ang. Control System)** - system, który rozwiązuje **Problem Zarządzania Zarządzanego Systemu** za pomocą **Zamkniętej Pętli Sterowania**. W każdej iteracji **System Sterowania** analizuje **Sprzeżenie Zwrotne** (ang. *Control Feedback*) i **wnioskuje Akcję Sterującą**.
- **Zamknięta Pętla Sterowania** - nieskończona pętla, która reguluje stan **Zarządzanego Systemu**, iteracyjnie zbliżając jego **Aktualny Stan** do **Stanu Pożądanego**.
- **Akcja Sterująca (ang. Control Action)** - akcja wykonywana na **Zarządzanym Systemie**, która ma na celu przybliżenie go do **Stanu Pożądanego**.
- **Sprzeżenie zwrotne** (ang. *Control Feedback*) - reprezentacja **Aktualnego Stanu** wysyłana z (odbierana od) **Systemu Zarządzanego**.

W powyższej architekturze Luples pełni rolę **Systemu Sterowania**.

3.3.6. Agenci Translacyjni

Z racji, że każdy system, bez żadnych modyfikacji (7) może wejść w rolę **Zarządzanego Systemu**, potrzebujemy warstwy integracji między **systemem zarządzanym** a Luples, podobna do koncepcji "API Broker" w architekturze ENI (rys. 2.3). Tak narodziła się koncepcja **Agentów Translacyjnych** (ang. *Trasnlation Agents*).

W każdym **wdrożeniu Luples**, **użytkownik** musi stworzyć **Agentów Translacyjnych**.

Z punktu wiedzenia komunikacji, każdego **Agenta Translacyjnego** możemy podzielić na dwie części:

- Część komunikująca się z **Systemem Zarządzanym**. Jest to zewnętrzna względem Luples i nie podlega żadnej specyfikacji.

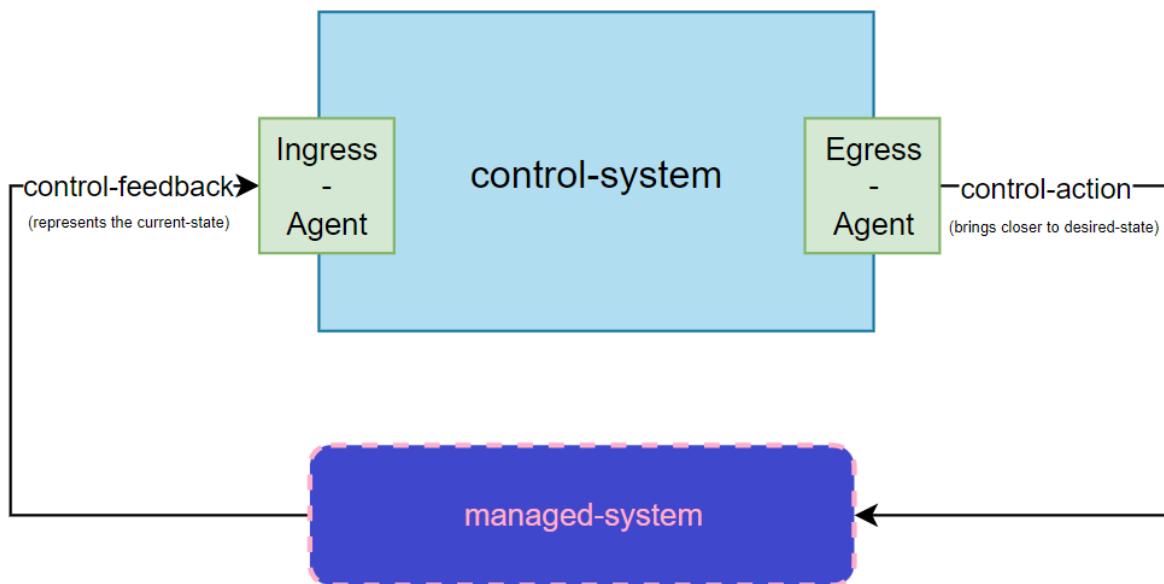
3. Architektura

- Część komunikująca się z Lupus. Musi być zgodne z jednym z interfejsów Lupus. **Lupin** lub **Lupout**.

Mamy dwóch **Agentów Sterowania**, jeden do komunikacji przychodzącej (ang. *Ingress*) i jeden do wychodzącej (ang. *Egress*).

W każdej iteracji pętli zadaniem Agenta Ingress jest odbieranie/zbieranie **Sprzężenia Zwrotnego** z **Systemu Zarządzanego** i translacja go na format zrozumiały przez Lupus za pomocą **Interfejsu Lupin**. Z kolei zadaniem **Agenta Egress** jest odbieranie **final-data'y** i translacja jej na Ackje Sterowania, która później zostaje wysłana do (lub przeprowadzona na) Systemie Zarządzanym.

Architektura wraz z wprowadzeniem **Agentów Translacji** ukaza jest na rysunku 3.2.



Rysunek 3.2. Architektura referencyjna z agentami translacji

Specyfikacja interfejsów **Lupin** oraz **Lupout** zawarta jest w załączniku //TODO.

3.3.7. Workflow pętli

Kiedy Lupus otrzyma **Aktualny Stan** poprzez interfejs **Lupin**, rozpoczyna się **Workflow Pętli**, które ma na celu dostarczać **Logikę Pętli** (ang. *Loop Logic*) i składa się z **Elementów Pętli** (ang. *Loop elements*).

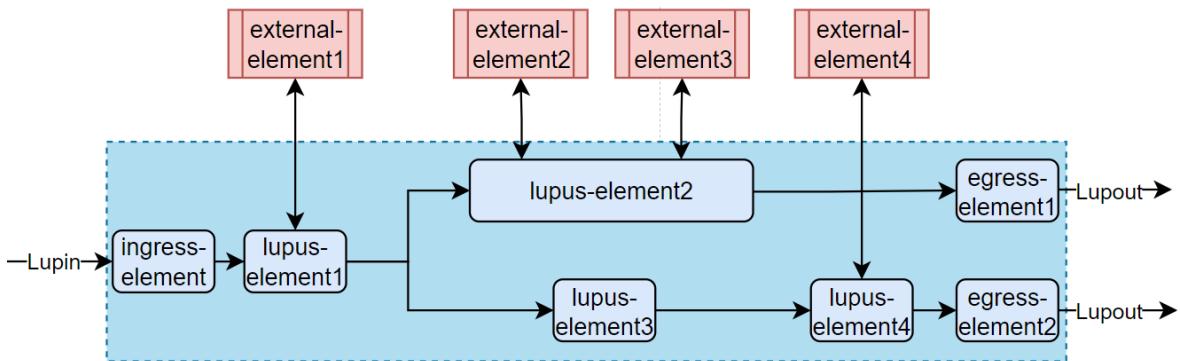
Elementem pętli może być zarówno:

- **Element Lupus**, który działa w warstwie sterowania Kubernetes a jego misją jest wykonywać *Workflow Pętli*
- referencja do **Elementu Zewnętrznego**, który działa poza warstwą sterowania Kubernetes a jego misją jest wykonywać **Część Obliczeniową Logiki Pętli**

Workflow Pętli jest wyrażany w **LupN**, specjalnej notacji do opisywania workflow pętli, której dokładna specyfikacja znajduje się w załączniku //TODO.

Przykładowe **workflow pętli** pokazano na rysunku 3.3

- Niebieskie, zaokrąglone prostokąty reprezentują **Elementy Lupus**,
- Czerwone prostokąty oznaczają **Elementy Zewnętrzne**.



Rysunek 3.3. Przykładowe workflow pętli

- Niebieski obszar wyznaczony linią przerywaną wskazuje elementy działające w warstwie sterowania Kubernetes.
- Wyróżniono elementy Lupus odpowiedzialne za ingress i egress.

Elementy Zewnętrzne to zazwyczaj serwery HTTP (w szczególności serwery Open Policy Agent).

Jak można zauważyć, jeden **Element Lupus** może komunikować się z żadnym, jednym lub wieloma elementami zewnętrznymi, a liczba ta może się różnić w każdej iteracji pętli).

3.3.8. Dane i Akcje

Dane (ang. *data*) to nośnik informacji w ramach jednej iteracji pętli w postaci JSON. Na wejściu **Elementu Ingress** reprezentują one **aktualny stan** (ang. *current state*) **Systemu Zarządzanego**. Następnie, w trakcie iteracji pętli, to projektant decyduje jakie informacje będą przenosić. Zazwyczaj są to informacje związane z **logiką pętli** takie jak wejścia (ang. *inputs*) do **Elementów Zewnętrznych** oraz ich odpowiedzi. Na końcu iteracji, gdy **dane** trafiają do **Agenta Egress** muszę reprezentować **Akcję Sterowania**.

Działanie pojedynczego **Elementu Lupus** jest opisane poprzez **Workflow Akcji**. **Akcja** wykonują różne operacje na **danych**. Istnieje wiele rodzajów akcji, a kluczowym typem akcji jest "Send", która pozwala komunikować się **Elementowi Lupus z Elementem Zewnętrznym**. Inne typy akcji służą organizacji danych.

//TODO daj tu przykładowe workflow akcji

Pełna specyfikacja **Danych** znajduje się w //TODO, zaś pełna specyfikacja **Akcji** w załączniku //TODO.

3.3.9. Open Policy Agent

Open Policy Agent (OPA) to otwartoźródłowe narzędzie przeznaczone do definiowania, egzekwowania i zarządzania politykami w systemach oprogramowania. Systemy informatyczne odpytują serwery OPA podczas wykonywania operacji, które wymagają decyzji (decyzji dostępu, konfiguracji czy zachowania aplikacji). Dzięki takiemu podejściu można oddzielić logikę podejmowania decyzji od kodu aplikacji, co zwiększa modularność, wprowadza centralny punkt zarządzania politykami oraz ułatwia utrzymanie systemów.

3. Architektura

Polity w OPA definiowane są w języku Rego. Jest to deklaratywny język stworzony specjalnie na potrzeby OPA. Pozwala na tworzenie złożonych reguł i logiki decyzji.

//TODO official site

Listing 1. Przykładowy kod rego

```
allow {  
    input.user.role == "admin"  
}  
  
allow {  
    input.user.role == "user"  
    input.action == "read"  
}
```

Open Policy Agent jest rekomendowanym **Elementem Zewnętrzny** dla Lupus.

3.4. Instrukcja Użycia

Niniejsza podsekcja prezentuje krótką instrukcję użycia platformy Lupus. **Użytkownikiem** Lupus może stać się dowolna organizacja bądź pojedyncza osoba. Ważne jest aby zespół użytkownika posiadał kompetencje z zakresu tworzenia oprogramowania. W zespole **Użytkownika** wyróżniamy rolę **Designera**, który jest odpowiedzialny jedynie za projektowanie i wyrażanie **Workflow Pętli**, i niekoniecznie musi posiadać umiejętności techniczne.

Kiedy dany **Użytkownik** planuje użyć Lupus jako **Systemu Kontroli** do rozwiązywania **Problemu Zarządzania** w swoim **Systemie Zarządzanym**, powinien:

1. Zainstalować Lupus w swoim klastrze
2. Zintegrować **Zarządzany System** z Lupusem poprzez implementację **Agentów Translacji**
3. Zaplanować **Workflow Pętli** (w dowolnym narzędziu)
4. Przygotować **Elementy Zewnętrzne**
5. Wyrazić **Workflow Pętli** w **LupN**
6. Zaaplikować pliki manifestacyjne zawierające kod **LupN** w klastrze

Podjęcie przez użytkownika takich akcji nazywamy pojedynczym **Wdrożeniem Lupus**.

3.4.1. Instalacja Lupus

Lupus jest zaimplementowany jako Niestandardowe Zasoby (ang. *Custom Resources*) w Kubernetes. Instalacja polega na zainstalowaniu tych zasobów w swoim klastrze.

Pełna dokumentacja tego procesu znajduje się w załączniku //TODO

3.4.2. Implementacja Agentów Translacyjnych

To użytkownik podczas **Wdrożenia** jest odpowiedzialny za implementację agentów translacji. Tylko użytkownik zna specyfikę swojego Systemu Zarządzanego. Stąd w zespole użytkownika potrzebne są umiejętności programistyczne.

Podczas implementacji należy kierować się specyfikacją interfejsów Lupin oraz Lupout zawartą w załączniku //TODO.

3.4.3. Planowanie Workflow Pętli

Workflow pętli powinno wyrazić logikę rekoncylacji, czyli w każdej iteracji doprowadzić System Zarządzany do Stanu Pożdanego. Użytkownik w tym kroku jedynie modeluje wysokopoziomowo workflow pętli, rysując np. jego diagram. Dopiero takie spojrzenie podpowie użytkownikowi jakich elementów zewnętrznych potrzebuje.

3.4.4. Przygotowanie Elementów Zewnętrznych

Elementem Zewnętrznym może być dowolne oprogramowanie, z którego **Designer** ma chęć włączyć w **workflow pętli**. Mogą to być już gotowe systemy (np. sztucznej inteligencji), ale równie dobrze użytkownik może stworzyć oprogramowanie samemu. Ważne jest to, aby wystawały one jakiś sposób komunikacji. Na razie jedynym wspieranym przez Luples sposobem jest komunikacja HTTP. Użytkownik musi umożliwić, więc komunikację tego typu z elementem.

Rekomendowanym przez Luples typem Elementu Zewnętrznego jest serwer Open Policy Agent. Przygotowanie w tym wypadku polega na uruchomieniu takiego serwera oraz wgraniu mu odpowiednich polityk. Użytkownik jednakże może wydewelopować dowolny serwer HTTP.

Ostatnim możliwym elementem zewnętrznym są **Funkcje Użytkownika**. Funkcje w kodzie kontrolera zasobu **Luples Element**, które są definiowane przez użytkownika. Stanowią one alternatywę dla serwerów HTTP, gdy specjalne wdrażanie takowych może się okazać zbyt kosztowne. Przykładowo jeśli logika wykonywana przez dany **Element Zewnętrzny** miała by mieć tylko kilka linijek kodu, dużo łatwiej użyć **Funkcji Użytkownika**. Ich dokładna specyfikacja znajduje się w załączniku //TODO.

3.4.5. Wyrażenie workflow pętli

Gdy już całe workflow pętli oraz elementy zewnętrzne są gotowe czas wyrazić jej workflow w notacji **LupN**. Za jej pomocą wyraża się workflow pętli jako zbiór **Elementów Luples**, połączenia między nimi oraz specyfikacja każdego z nich.

3.4.6. Aplikacja plików manifestacyjnych

Aby stworzyć pętle opisaną przez kod LupN w pliku manifestacyjnym zasobu Master należy wykonać komendę z listingu 2, która tworzy instancję obiektu API typu Master. Kontroler tego zasobu stworzy odpowiednie obiekty API typu Element według specyfikacji w pliku LupN. Podczas iteracji pętli, kontroler każdego elementu interpretuje ich specyfikacje wykonując odpowiednie **akcje na danych**. **Workflow Akcji** również specyfikowane jest w notacji **LupN**, której specyfikacja znajduje się w załączniku //TODO.

Listing 2. Stworzenie zasobu Master

```
kubectl apply -f <nazwa_pliku>
```

4. Implementacja

4.1. Wstęp

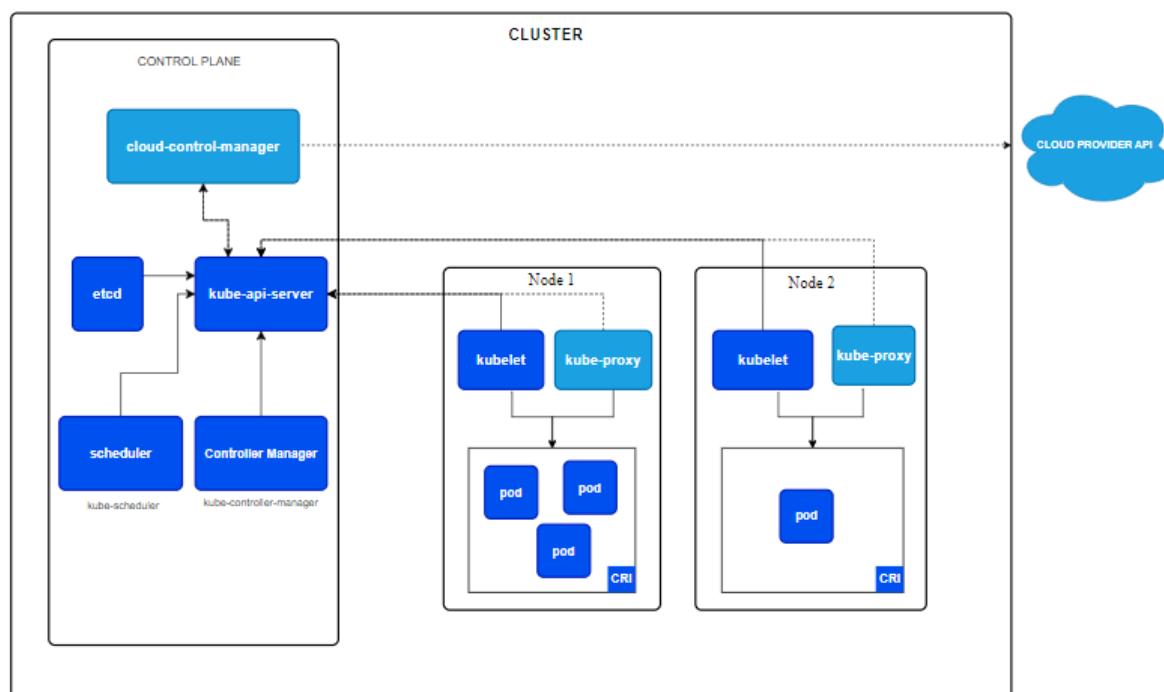
Sekcja ta opisuje wydzieloną z architektury implementację Luples. Pierwsza jej części opisuje kluczowe mechanizmy Kubernetes, za pomocą których zaimplementowano Luples. Druga część opowiada o decyzjach podjętych podczas developmentu platformy. To tu objawia się badawcza natura pracy.

4.2. Mechanizmy Kubernetes stojące za Luples

Niniejszy dokument zakłada znajomość czytelnika platformy Kubernetes na podstawowym poziomie.

4.2.1. Kontroler

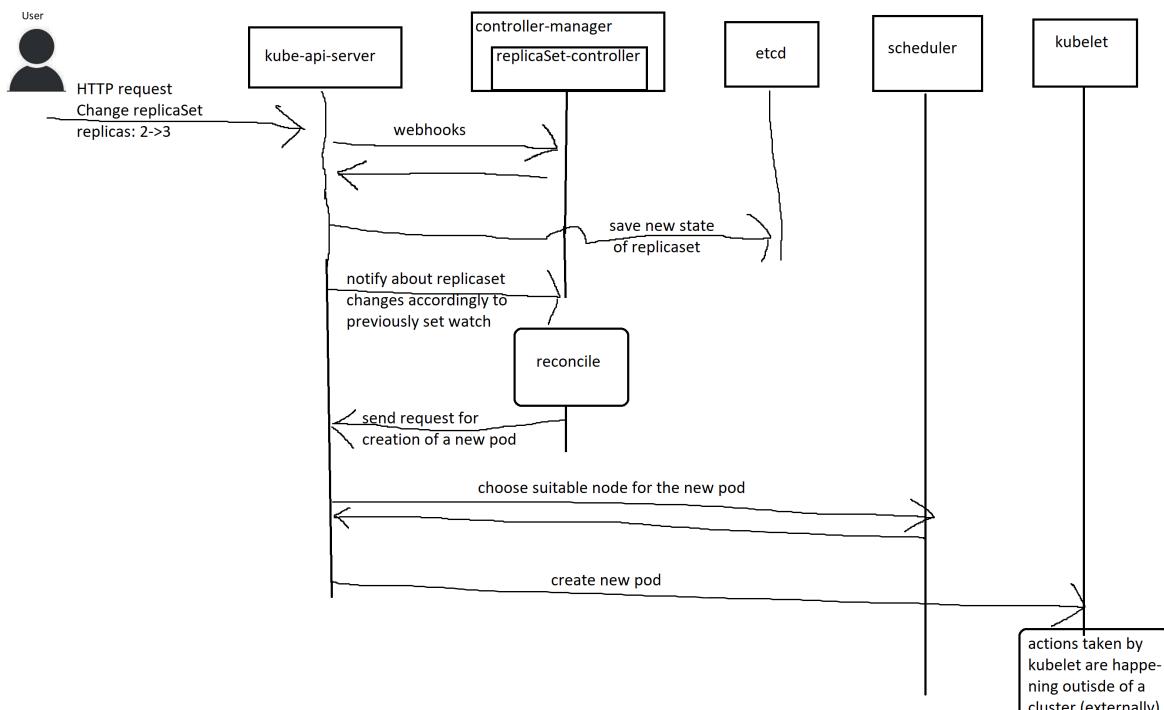
Działanie Kubernetes opiera się na zamkniętej pętli sterowania [odnośnik do artykułu]. **Aktualny Stan** systemu zapisany jest w bazie etcd. **Stan Pożądany** wyrażony jest poprzez pliki manifestacyjne. Każdy *Obiekt API* posiada swoją sekcję ‘spec’, to ona definiuje pożądanego stan danego obiektu. Każdy obiekt ma swój kontroler, który rekoncyluje jego aktualny stan do stan pożądanego. Kontroler jest to proces działający w warstwie sterowania Kubernetes. Każdy typ (ang. *Kind*) wbudowanych zasobów (ang. *built-in resources*) ma swój kontroler stworzony przez zespół Kubernetes. Kontrolery każdego typu zasobu działają w podzie ‘kube-controller-manager’.



Rysunek 4.1. Architektura Kubernetes

Flow pracy kontrolera pokazano na rysunku 4.2. Gdy ‘kube-api-server’ otrzyma żądanie zmiany danego obiektu API zanim ‘kube-api-server’ zleci utrwalenie tych zmian wykona

tzw. *webhooki* do kontrolera danego obiektu. Webhooki nie są jednakże istotne z punktu widzenia niniejszej pracy dyplomowej⁵. Następnie, gdy doszło do zmian w obiekcie, kontroler zostaje o tym poinformowany. Jego misją jest rekoncylacja, czyli porównanie aktualnego stanu obiektu ze stanem pożądany. Kontroler zawiera w sobie logikę rekoncylacji, która przybliży oba stany do siebie. Wykonanie logiki wiąże się często z przeprowadzeniem różnych akcji przez kontroler w innych częściach klastra.



Rysunek 4.2. Flow pracy kontrolera

4.2.2. Zasoby niestandardowe

Zasoby niestandardowe (ang. *Custom Resources*) rozszerzają wbudowane zasoby (ang. *built-in resources*) o niestandardowe, zdefiniowane przez użytkownika. Najczęściej tworzone są w celu zarządzania konfiguracją skomplikowanych lub stanowych aplikacji.

Czasami wbudowane typy zasobów (jak Pody, Wdrożenia (ang. *Deployments*), Serwisy (ang. *Services*)) nie są dla nas wystarczające, dlatego Kubernetes udostępnia możliwość rejestrowania nowych typów. Aby tego dokonać należy stworzyć plik manifestacyjny YAML typu *Custom Resource Definition* a następnie zaaplikować go w klastrze. Po pomyślnej operacji, będzie można tworzyć obiekty nowego typu.

4.2.3. Operatory

Mechanizm ten służy do rozszerzania możliwości Kubernetes. Kiedy tworzymy Zasoby Własne (ang. *Custom Resources*), możemy również implementować dla nich kontrolery. Takie podejście nazywane jest "wzorem operatora" ("operator-pattern"). Często takie kontrolery nazywamy po prostu "operatorami". Nazwa "operator" wywodzi się z idei, że taki

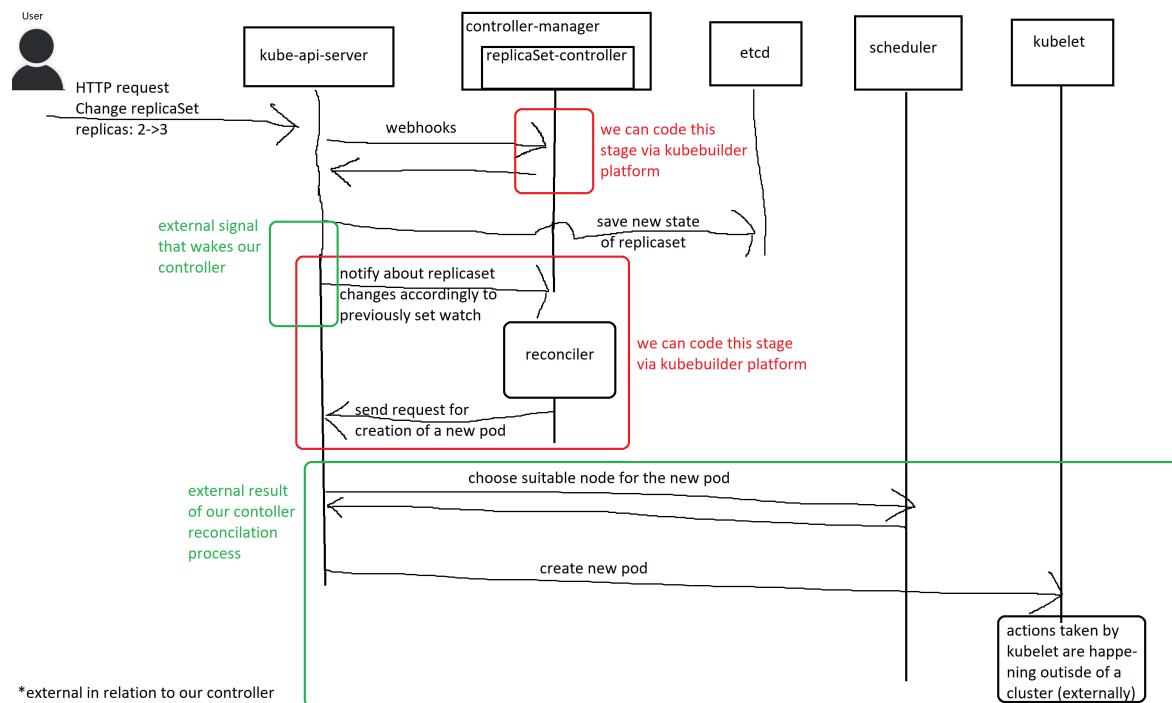
⁵ Dla zainteresowanych tematem: link

4. Implementacja

kontroler zazwyczaj zastępuje rzeczywistego operatora (człowieka), który zarządzałby aplikacją (dla której wdrożenie wymagało zdefiniowania Zasobu Własnego).

4.2.4. Kubebuilder

Kubebuilder jest frameworkm programistycznym do tworzenia Zasobów Własnych oraz ich operatorów. Pozwala na to, aby zaprogramować sekcje zaznaczone na rysunku 4.3.



Rysunek 4.3. Zaznaczenie elementów dających się zaprogramować podczas flow pracy kontrolera

Bibliografia

- [1] ETSI GR ENI 017 - "Experiential Networked Intelligence (ENI); Overview of Prominent Closed Control Loops Architectures", Dostęp zdalny (24.01.2025): https://www.etsi.org/deliver/etsi_gr/ENI/001_099/017/02.02.01_60/gr_ENI017v020201p.pdf, 2024.

Wykaz symboli i skrótów

ETSI – European Telecommunications Standards Institute

ENI – Experiential Networked Intelligence

Spis rysunków

2.1 Pracowniczki warszawskiej centrali telefonicznej z końca lat 20. XX wieku	9
2.2 Dwa stany systemu: przed, oraz po realizacji zgłoszenia o treści: "D" chce zadzwonić do "F"	10
2.3 Wysokopoziomowa architektura funkcjonalna ENI	12
2.4 Architektura pętli OODA	13
2.5 Architektura pętli MAPE-K	13
2.6 Architektura pętli FOCALE	14
2.7 Architektura pętli COMPA	14
2.8 Układ automatycznej regulacji (UAR)	15
2.9 Wykresy pomiaru, punktu odniesienia oraz wymuszenia w czasie	16
3.1 Architektura referencyjna dla Luples	19
3.2 Architektura referencyjna z agentami translacji	20
3.3 Przykładowe workflow pętli	21
4.1 Architektura Kubernetes	24
4.2 Flow pracy kontrolera	25
4.3 Zaznaczenie elementów dających się zaprogramować podczas flow pracy kontrolera	26

Spis tabel

Spis załączników

1. Definicje	28
------------------------	----

Załącznik 1. Definicje

Architektura – Zbiór reguł i metod opisujących funkcjonalność, organizację oraz implementację systemu.

Workflow – Sekwencja połączonych węzłów, czasami zależnych warunkowo, która realizuje określony cel. Zazwyczaj workflow definiuje się w celu organizacji pracy

Wnioskowanie (ang. reasoning) – Proces, w którym system wyciąga logiczne wnioski z dostępnych danych i wiedzy

Kognitywność (ang. cognition) – Proces rozumienia danych oraz informacji w celu produkcji nowych danych, informacji oraz wiedzy

System kognitywny – System, który uczy się, wnioskuje oraz podejmuje decyzje w sposób przypominający ludzki umysł