**Uniwersytet Warszawski**
Wydział Matematyki, Informatyki i Mechaniki

**Cezary Kaliszyk**
Nr albumu: 189400

# Moduł komunikacyjny systemu optymalizacji i personalizacji serwisów WWW SIE

**Praca magisterska**
**na kierunku INFORMATYKA**
**w zakresie OPROGRAMOWANIA I METOD INFORMATYKI**

Praca wykonana pod kierunkiem
**dra Aleksego Schuberta**
Instytut Informatyki

Luty 2005

## Oświadczenie kierującego pracą

Oświadczam, że niniejsza praca została przygotowana pod moim kierunkiem i stwierdzam, że spełnia ona warunki do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

## Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

## Streszczenie

Niniejszy dokument jest opisem modułu komunikacyjnego dla systemu *Site Improving Engine* optymalizującego i personalizującego serwisy WWW. Moduł ten został stworzony jako praca magisterska. Nowy moduł komunikacyjny dzięki asynchronicznej komunikacji zwiększa bezpieczeństwo systemu oraz pozwala na uzyskanie wysokiej wydajności, nawet przy dużym obciążeniu serwisu. System *SIE* jest podstawą projektów przystosowujących strony WWW do potrzeb indywidualnego użytkownika (ang. *adaptive web*). Programista, pisząc własną wtyczkę, dostaje od systemu obsługę wszelkich niezbędnych protokołów sieciowych, obsługę pracy w klastrze, identyfikację użytkowników, interpretację dokumentów HTML, bazę danych i obsługę błędów. System jest napisany w funkcyjnym języku OCaml, co dodatkowo ułatwia pisanie nowych wtyczek, przy zachowaniu jednocześnie wysokiej wydajności.



## Słowa kluczowe

samoadaptująca się sieć, OCaml, WWW, serwis WWW, ergonomia, HTML, HTTP, asynchroniczne I/O

## Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 — Informatyka

## Klasyfikacja tematyczna

H — Information Systems
H.3 — Information Storage and Retrieval

# Spis treści

# Rozdział 1

# Wprowadzenie

## 1.1. Problemy reprezentacji informacji w Internecie

W dzisiejszym świecie występuje niespotykany dotąd rozwój technologii związanych z udostępnianiem i przetwarzaniem informacji. W związku z oferowaną przez Internet globalizacją niewątpliwie możliwość dostępu do poszukiwanej informacji jest znacznie większa, często informacje nie są dostępne lub są trudno dostępne w inny sposób. Jednak ogrom informacji z którymi styka się współczesny użytkownik sieci, powoduje że czas potrzebny na znalezienie poszukiwanej informacji, a przez to koszt wyszukania informacji istotnie rośnie.

Tworzone są różne serwisy mające na celu grupowanie informacji. Coraz częściej jednak zdarza się, że ilość informacji przekracza możliwości układania struktury stron przez architektów serwisów.

Internetowe serwisy starają się tworzyć jak najbardziej klarowną strukturę łącz na swoich stronach, udostępniają tworzenie list ulubionych stron użytkowników, a także często udostępniają mapy serwisów pokazujące zhierarchizowaną i logicznie uporządkowaną strukturę serwisu.

Każda sztywnie stworzona struktura jest jednak tworzona dla przeciętnego użytkownika. Nie może brać pod uwagę nietypowych preferencji i zazwyczaj nie uwzględnia rzadko poszukiwanych informacji, gdyż utrudniałyby one używanie przeciętnemu użytkownikowi.

## 1.2. Jak System SIE wychodzi im naprzeciw

System *SIE* został napisany jako rozwiązanie powyższych problemów poprzez zapamiętywanie kolejno wykonywanych czynności przez użytkownika oraz następnie analizę wykonywanych działań na stronach internetowych. Dzięki temu może oferować na podstawie zebranych informacji strukturę serwisu dopasowaną do potrzeb indywidualnego użytkownika.

*Site Improving Engine* optymalizuje i personalizuje dynamicznie strukturę przekazywanych danych. Napisane dla systemu *SIE* moduły oferują optymalizację struktury linków indywidualnego serwisu poprzez podpowiadanie klientowi linków do stron, których z największym prawdopodobieństwem szuka.

Dodatkowo, system oferuje użytkownikom wyszukiwarkę (dobierającą strony także na zasadzie indywidualnych preferencji) oraz możliwość nagrywania i odtwarzania przebiegów sesji. Sesje mogą być odtwarzane wielokrotnie, równolegle oraz mogą być parametryzowane informacjami, które mają być wpisywane do formularzy.

## 1.3. Istniejący moduł komunikacyjny

System *SIE* w pierwotnej wersji był pisany jako próba zaimplementowania pewnej liczby pomysłów z dziedziny *Adaptive Web*, jednak nie miał być systemem ostatecznym, lecz raczej rozwiązaniem typu *proof of concept*. Z tego powodu moduł komunikacyjny będący główną częścią systemu *SIE* nie brał pod uwagę bardzo istotnych dla implementacji dużych serwisów WWW cech takich jak wydajność i bezpieczeństwo.

Moduł komunikacyjny był napisany z użyciem synchronicznego wejścia-wyjścia. Dla każdego połączenia tworzony był nowy wątek systemu, przez co możliwość jego skalowalności była istotnie ograniczona. System w jego pierwotnej wersji [1], działając jako pośrednik HTTP przed serwerem, spowalniał kilkakrotnie działanie serwisu. Do systemu zostały następnie dodane mechanizmy przyspieszające jego działanie w przypadku, gdy część informacji nie jest potrzebna [2]. Jednak ciągle nie było to wystarczające do uruchomienia systemu na prawdziwym serwisie o dużym obciążeniu.

Kolejnym problemem poprzedniego modułu był brak zapewniania przez niego bezpieczeństwa. Problemy z działaniem podczęści systemu powodowały jego całkowite wyłączanie, a przez to niedostępność całego serwisu. Mimo że system *SIE* był uruchomiony jako pośrednik HTTP, jego niedostępność całkowicie odcinała dostęp do znajdującego się za nim rzeczywistego serwera, posiadającego oryginalną wersję stron.

Dodatkowym problemem istniejącego systemu była istotna złożoność kompilacji oraz konfiguracji całego systemu. Często potrzebna była ręczna edycja różnych plików `makefile` rozrzuconych po katalogach. Zmienne konfiguracyjne podfragmentów systemu często znajdowały się w plikach źródłowych. Zmienne znajdowały się w różnych plikach, a ich format oraz wczytywanie było zaimplementowane przez części systemu oddzielnie i niezależnie. Oprócz zmiennych w plikach konfiguracyjnych część była także dostępna tylko jako opcje linii poleceń.

System zależał od wielu rzadko spotykanych bibliotek. W szczególności do jego konfiguracji i działania wymagane było uruchomienie bazy danych Postgres [32].

## 1.4. Nowy moduł komunikacyjny

Nowy moduł komunikacyjny jest napisany z myślą o dużej wydajności dzięki korzystaniu z mechanizmów asynchronicznej obsługi wejścia-wyjścia. W tym celu został napisany ogólny moduł z funkcjami obsługi sieciowej, zaimplementowany na dwa sposoby:

- Wersja korzystająca z modułu Unix biblioteki standardowej OCamla. Wersja ta korzysta z komunikacji synchronicznej i działa na wszystkich platformach na których działa OCaml (w szczególności pod systemem MS Windows odwołania do systemu są wykonywane za pomocą zestawu narzędzi Cygwin).
- Wersja korzystająca z biblioteki libevent [30] udostępniającej niezależny od platformy dostęp do oferowanego przez nią mechanizmu asynchronicznego wejścia-wyjścia. Biblioteka ta jest w stanie korzystać z takich mechanizmów jak `EPoll`, `KQueue` [31], `RTSigs` oraz inne, przy czym niezależnie od dostępnego na danej platformie mechanizmu, biblioteka ta korzysta z niego w sposób bardzo efektywny [34, 35].

Dodatkowo w skład nowego modułu komunikacyjnego wchodzi *WATCHDOG*, czyli mechanizm monitorowania elementów klastra, który w przypadku nie działania któregoś z procesów przetwarzających przenosi żądania przez niego obsługiwane najpierw na pozostałe elementy klastra, a jeśli i to zawiedzie, to bezpośrednio do serwera WWW.

System plików `makefile` został przepisany w taki sposób, aby wszystkie zmienne konfiguracyjne dla kompilacji były w jednym pliku w katalogu głównym. Ponadto zależności pomiędzy plikami

są automatycznie wyliczane, a kompilacja może zależeć tylko od biblioteki libevent (jej używanie nie jest obowiązkowe).

Wszystkie zmienne konfiguracyjne systemu są umieszczone w jednym pliku w katalogu głównym. Ponadto wszystkie programy posiadają te same opcje konfiguracyjne dostępne z linii poleceń. W przypadku, kiedy wartość danej zmiennej dostępna jest zarówno z pliku, jak i z linii poleceń, te drugie są traktowane jako ważniejsze.

## 1.5. Wykorzystane narzędzia i protokoły

Protokoły komunikacyjne:

- TCP/IP — podstawowe protokoły komunikacyjne sieci Internet. System *SIE* korzysta z wbudowanych w system operacyjny mechanizmów obsługi tego protokołu albo za pomocą modułu Unix [28] biblioteki standardowej OCamla albo za pomocą biblioteki do komunikacji asynchronicznej libevent [30].

- HTTP — protokół przesyłania stron WWW opisany w RFC 2616 [25]. Jest to protokół wyższego poziomu niż TCP/IP. System *SIE* zawiera własny parser i generator wierszy nagłówków HTTP.

Języki:

- HTML — powszechnie stosowany język opisu stron WWW [24]. Pozwala umieścić na stronie tekst i grafikę oraz połączyć strony linkami.

- OCaml — język programowania funkcyjnego zawierający elementy obiektowe. System został w całości napisany w tym języku.

Narzędzia:

- OCamlDoc — narzędzie służące do automatycznego generowania dokumentacji z kodu języka OCaml. Jest wykorzystywany do generowania dokumentacji kodu systemu.

- ZeusBench, ApacheBench — narzędzia służące do mierzenia przepustowości serwerów WWW. Pozwalają na sprawdzanie średniej liczby stron obsługiwanych w danym czasie przez serwis. Potrafią one mierzyć przepustowość zarówno w przypadku żądań zadawanych sekwencyjnie jak i równolegle.

- Httperf — mierzy wydajność serwera WWW zadając zapytania z określoną częstością lub gęstością [37].

- Deadconn — program tworzący określoną liczbę połączeń do serwera symulujących połączenia wolne lub martwe. Dla każdego połączenia wysyłana jest minimalna ilość danych do serwera, co powoduje, że serwer musi trzymać informację o połączeniu w oczekiwaniu na dalsze dane uzupełniające zapytanie. Ponadto w przypadku mechanizmów zapytywania o dane na deskryptorach (`select`, `poll`, ...), niezbędne jest przekazywanie wszystkich tych deskryptorów do zapytań, co zwiększa ich złożoność.

## 1.6. Moje doświadczenia

Podczas pisania pierwszej wersji systemu *SIE* kierowałem grupą piszącą główny moduł obsługujący protokoły sieciowe oraz komunikację.

Istotna część istniejącego modułu komunikacyjnego, obsługi protokołu HTTP oraz interfejs programistyczny dla modułów zostały napisane przeze mnie.

Podczas pisania licencjatu jako zespołowy projekt programistyczny, grupa pisząca system liczyła szesnaście osób. Połowa z nich po raz pierwszy zetknęła się z językiem programowania, w którym system był pisany. Dlatego istotne było maksymalne uproszczenie dostępnego dla modułów *SIE* interfejsu programistycznego. Zdecydowaliśmy się na napisanie *SIE* w architekturze synchronicznej z jednym wątkiem obsługującym jedno żądanie klienta.

Grupa szesnastoosobowa została podzielona na cztery grupy czteroosobowe, z których każda miała zaimplementować wydzielony fragment systemu. Ponieważ grupy podczas pisania swoich części nie komunikowały się często, każda z nich przechowywała niezależnie takie rzeczy, jak zmienne konfiguracyjne.

Ponadto system pisany był jako *proof of concept* pewnej liczby prac z dziedziny „samoadaptującej się sieci" (ang. *adaptive web*). Ponieważ dziedzina ta jest dziedziną dosyć nową, często w trakcie pisania znajdowaliśmy dodatkową funkcjonalność, którą można dodać lub uogólnić pewną liczbę wykonywanych czynności. Spowodowało to, że w trakcie pisania pierwszej wersji systemu, miało miejsce dużo zmian, w jego architekturze.

## 1.7. Mój wkład

Pewna liczba osób rozwija *SIE*, zmiany które wykonałem osobiście w ramach pracy magisterskiej są następujące:

- Przepisałem system plików makefile oraz dodałem wszystkie niezbędne pliki do działania systemu configure.
- Napisałem główną część metod komunikacji korzystających z ocamlowej warstwy pośredniej dookoła biblioteki libevent.
- Przepisałem cały mechanizm komunikacji głównego procesu przetwarzającego, jego główną pętlę oraz zarządzanie wątkami i muteksami.
- Przepisałem istotną część interfejsu do centralnej bazy danych, tak aby działała z mechanizmami asynchronicznymi.
- Zaimplementowałem mechanizm *WATCHDOG*.

## 1.8. Zawartość dokumentu

W rozdziale 2 zostały opisane założenia, którymi kierowano się przy projektowaniu systemu *SIE* oraz technologie, których użyto do osiągnięcia tych wymagań.

W rozdziale 3 opisana została architektura całego systemu, podział na części składowe oraz ich zadania. Opisane zostały mechanizmy, które służą do osiągnięcia wymaganej funkcjonalności.

W rozdziale 4 zawarty został opis możliwych zastosowań systemu, jak również opis testowych i konkretnych użyć systemu *SIE*. Przedstawione są wyniki stosowania systemu wraz ze statystykami odwiedzin serwisów oraz porównaniem różnych własności komunikacji, w zależności od wersji serwisu *SIE*.

W dodatku A, załączony został podręcznik administratora systemu, opisujący dokładnie procedurę kompilacji i instalacji systemu, jak również działania potrzebne do administrowania nim.

Na koniec, w dodatku B, załączony został słownik wyjaśniający terminy związane z oglądaniem stron WWW, tak jak są rozumiane w tym dokumencie.

# Rozdział 2

# Zasada działania

## 2.1. Własności systemu

Podczas projektowania systemu *SIE* zostały wzięte pod uwagę następujące wymagania dotyczące jego funkcjonalności:

- Analiza działań użytkownika — wybieranych przez niego stron z serwisu WWW, informacji zawartych na tych stronach, czasu oglądania każdej strony, kolejności klikanych linków, etc.

- Dynamiczne dodawanie spersonalizowanej struktury linków — przy przesyłaniu każdej strony nasz system stara się „domyśleć", jakie strony użytkownik będzie chciał zobaczyć w następnej kolejności i dodaje do nich linki.

- Udostępnienie wyszukiwarki — wyszukiwarka pokazuje strony zawierające słowa kluczowe podane przez użytkownika. Stara się przeanalizować, które strony będą ciekawsze dla konkretnego klienta.

- Przezroczystość dla użytkownika — użytkownik nie musi instalować specjalnego oprogramowania, używa zwykłej przeglądarki i łączy się z serwerem WWW. System przechwytuje komunikację w obie strony pomiędzy serwerem a klientem i wzbogaca strony WWW. Gdyby nie nowe linki, użytkownik w ogóle by nie wiedział o pośrednictwie naszego systemu.

- Skalowalność — poprzez zainstalowanie systemu na większej liczbie komputerów można prawie dowolnie zwiększać wydajność naszego systemu.

- Wysoka wydajność pracy — nasz system, gdy jest zainstalowany na wystarczającej liczbie komputerów, w sposób niezauważalny opóźnia otrzymanie strony WWW przez użytkownika.

- Niezawodność — w system są wbudowane mechanizmy pozwalające na wyłączenie całości lub części systemu, która źle działa (*WATCHDOG* monitoruje i ew. wyłącza elementy systemu lub cały system automatycznie).

- Modularność i łatwość modyfikacji — system został podzielony na dobrze wyspecyfikowane części z ustalonym interfejsem. Można bez problemu modyfikować oraz dopisywać nowe moduły poprawiające strony WWW.

## 2.2. Zaimplementowane technologie

### 2.2.1. Identyfikacja użytkowników

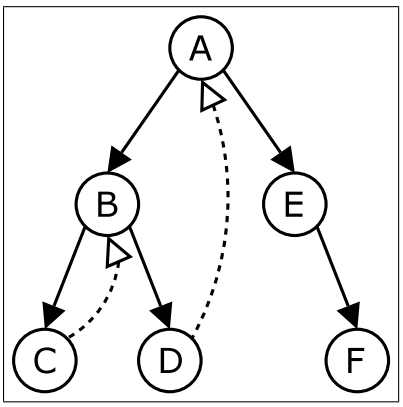
*SIE* działając jako pośrednik HTTP, niewidoczny dla klienta, przechwytuje wszystkie zapytania HTTP skierowane do serwera WWW i samo generuje odpowiedzi dla klienta. System, dzięki przechwytywaniu zapytań, może sam identyfikować użytkownika, aby następnie udostępnić tą informację modułom.

Na identyfikację użytkowników składają się 2 etapy:

- Podczas pierwszego zadanego zapytania przez klienta sprawdzane jest, czy ma on ustawione specjalne ciasteczko (ang. *cookie*) z identyfikatorem tego użytkownika w systemie. Jeśli tak, aktualna sesja klienta jest identyfikowana z tym właśnie użytkownikiem oraz wcześniejszą historią jego kliknięć i wybranymi przez niego preferencjami. W innym przypadku przydzielany jest nowy unikatowy identyfikator.
- Kiedy użytkownik kontynuuje klikanie linków w serwisie, ciasteczka nie są już potrzebne, gdyż system przepisuje linki i możliwa jest jednoznaczna jego identyfikacja dzięki samemu kliknięte mu linkowi.

### 2.2.2. Sesje kliknięć

*SIE* udostępnia wszystkim modułom historię kliknięć klienta w danej sesji w postaci drzewa kliknięć — to znaczy pamiętana jest nie tylko historia lecz także to z jakiej strony klient wybrał link. Użytkownik może wejść na stronę ze wszystkich stron posiadających odnośniki do danej. W szczególności korzystając z kilku okien lub zakładek przeglądarki użytkownik może wejść na tą samą stronę z różnych miejsc. Ponadto użytkownik może używać przycisku `Wstecz` przeglądarki lub zakładek.




Rysunek 2.1: Przykładowe drzewo pokazujące drogę użytkownika wewnątrz jego sesji. Linie ciągłe pokazują kliknięcie, natomiast linie przerywane pokazują naciśnięcia przycisku `Wstecz` w przeglądarce

*SIE* dzięki mechanizmowi przepisywania linków, potrafi rozpoznać, skąd nastąpiło kliknięcie i dostęp do takiej historii sesji klienta jest oferowany modułom.

### 2.2.3. Przepisywanie linków

We wszystkich stronach przesyłanych użytkownikowi, wszystkie linki zarówno wewnętrzne jak i do stron spoza serwisu są zmienione tak, aby jednoznacznie wskazywać użytkownika i jego sesję. Jednoznaczność osiągnięto za pomocą generowania 256-bitowych numerów i przechowywaniu ich w ta-

blicach haszujących. Tablice te są okresowo czyszczone, aby zapobiec mnożeniu się niepotrzebnych wpisów.

Do linków dodawane są następujące parametry:

- `SIE_SESSION` — Parametr identyfikujący sesję i użytkownika.
- `SIE_LINK` — Parametr identyfikujący konkretny link, którego klient użył, aby zapytać o stronę. Na podstawie tej informacji można odtworzyć oryginalny adres strony, na której był kliknięty link, jak i adres strony, do której link prowadzi.
- `SIE_ORIGINAL` — Parametr zawierający oryginalny link. Pozwala odtworzyć adres strony docelowej w przypadku gdyby oryginalny wpis z tablicy haszującej z linkami przepisanymi został wyczyszczony (na przykład link został zachowany jako zakładka w przeglądarce) lub kiedy odnosi się on do serwisu zewnętrznego.

Na przykład link:
`http://www.google.com`
zostaje przepisany na:
`http://sie.mimuw.edu.pl:8040?SIE_SESSION=EA755A6C&SIE_LINK=DC4A90A0&SIE_ORIGINAL=www.google.com`

### 2.2.4. Moduły

Częścią systemu, której funkcjonalność jest widoczna dla użytkowników, są moduły. Moduły zazwyczaj składają się z dwóch części: *OFFLINE* i *ONLINE*. Moduły *OFFLINE* dostają od systemu logi oraz możliwość iterowania po nich. Główną funkcjonalność system oferuje modułom *ONLINE,* które są ścisłą częścią systemu *ONLINE SIE*. Interfejs do modułów został tak przepisany, aby było możliwe ich dynamiczne ładowanie[1].

Moduł podczas rejestrowania się do *CZĘŚĆI ONLINE* podaje jej następujące parametry:

- Obsługiwane formaty zawartości (ang. *content types*), na przykład obsługiwany szczególnie `text/html` lub `image/png`.
- Wymagane wywołania (czyli obsługa zapytania od klienta do serwera lub obsługa odpowiedzi serwera).
- Priorytet (potrzebny do ustalenia kolejności, w jakiej wywołania modułów są wykonywane).
- Jeśli moduł obsługuje format zawartości `text/html`, specyfikuje także, czy wymaga sparsowanego drzewa HTML podczas modyfikacji odpowiedzi. Jeśli nie, moduł dostaje stronę w postaci pojedynczego napisu.

Komunikaty HTTP, które zawierają dokument HTML, są parsowane i deparsowane tylko w przypadku, gdy któryś z modułów aktualnie obsługujących dane żądanie wymaga parsowania. W innym przypadku wszystkie moduły dostają zawartość dokumentu HTML w postaci czystego tekstu.

Kiedy podczas obsługi żądania moduły są wołane przez *SIE,* dostają od systemu następujące informacje:

[1] To znaczy: każdy moduł, podczas inicjalizacji wywołuje jedną funkcję. Jednak aktualnie w standardowej dystrybucji OCamla możliwe jest dynamiczne ładowanie modułów tylko w wersji bytecodowej. W wersji skompilowanej wymaga to nakładki na OCamla, która na dzień dzisiejszy nie jest dostępna dla ostatniej wersji kompilatora, dlatego *SIE* nie korzysta z tego aktualnie [36].

- parametry zapytań o odpowiedzi HTTP,
- sparsowane drzewo HTML (jeśli jest to potrzebne, jak wyżej),
- informację o użytkowniku,
- drzewo kliknięć użytkownika w danej sesji. Zazwyczaj moduły korzystają ze ścieżki, czyli najkrótszej drogi od początku sesji do aktualnego węzła w drzewie.

### 2.2.5. Architektura klastrowa

System został napisany w taki sposób, aby było możliwe rozdzielenie głównych procesów obsługujących zapytania pomiędzy odrębne elementy klastra. Dzięki temu główna część systemu — czyli moduły mogą działać na innych komputerach, przez co możliwa jest większa skalowalność.

Dla synchronizacji danych została zapewniona centralna baza danych zawierająca dane, które muszą być współdzielone przez procesy obsługujące. Dodatkowo możliwa jest aktualizacja tych danych, po czym nowe informacje są aktualizowane w procesach przetwarzających.

Ponadto centralna baza umożliwia także przechowywanie i aktualizację konfiguracji na wszystkich komputerach klastra.

### 2.2.6. Bezpieczeństwo, WatchDog

Częścią składową systemu jest proces rozdzielający zapytania pomiędzy procesy przetwarzające. Proces ten, w zależności od numeru klienta, przydziela mu pewien proces przetwarzający. Przez cały czas trwania sesji użytkownika, ten sam proces przetwarzający obsługuje wszystkie jego zapytania.

Mimo, że *SIE* zostało napisane tak, aby nie miało błędów, nie jest możliwe całkowite zapewnienie bezpieczeństwa. W szczególności moduły, które są załadowanie do systemu, mogą powodować błędy lub się pętlić. Dlatego w proces ten wbudowany został także mechanizm *WATCHDOG* odpowiedzialny za dostępność całego systemu. Potrafi on wykrywać niedziałające procesy przetwarzające i wyłączać przesyłanie żądań do nich. Ponadto w przypadku kiedy cały system przestaje działać, proces ten potrafi wyłączać cały system i przesyłać żądania bezpośrednio de serwera WWW.

# Rozdział 3

# Opis architektury systemu

## 3.1. Całość systemu

System *SIE* dzieli się na dwie funkcjonalne części:

- *CZĘŚĆ ONLINE* — na którą składają się wszystkie czynności wykonywane w trakcie obsługi zapytań klienta i niezbędne do tego celu.
- *CZĘŚĆ OFFLINE* — w której wykonywane są obliczenia i analizy możliwe do wykonania a priori. Wyniki tych działań będą następnie wykorzystywane później w części *ONLINE*.

## 3.2. Część Online

Przeglądarka klient, który chce pobrać stronę, wysyła *zapytanie HTTP* do serwera WWW. System *SIE*, w sposób niewidoczny dla użytkownika, przechwytuje to zapytanie, zanim dotrze ono do serwera.

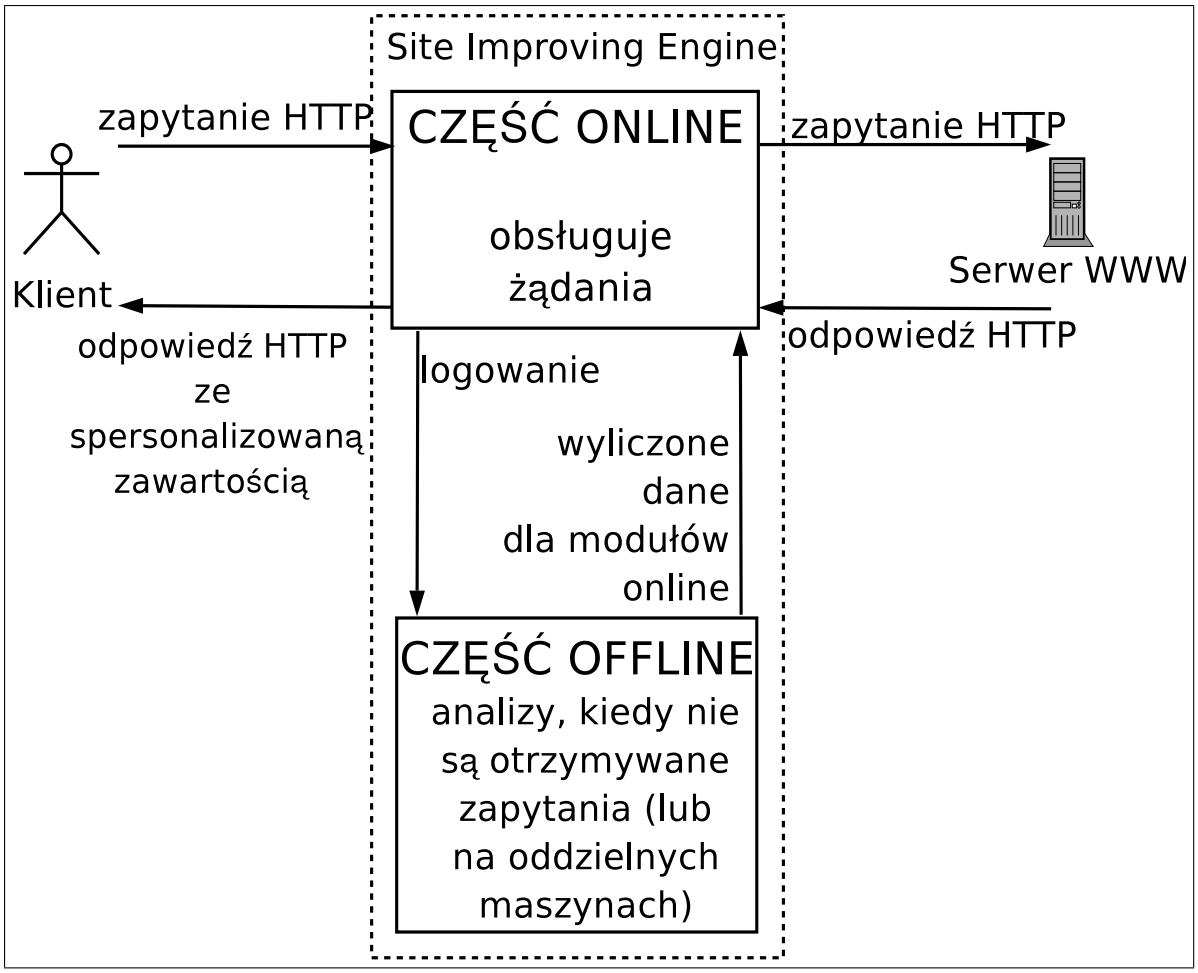
Zapytanie trafia najpierw do *REQUEST BROKERA*, który przekazuje je dalej do *BOKSA*. *BOKSÓW* może być kilka i są to równolegle działające aplikacje, każda na innym komputerze. Zastosowana została tutaj tzw. *architektura klastrowa*, czyli pracę wykonuje kilka równolegle działających maszyn, wykonujących dokładnie takie same operacje. Zwiększa to wydajność oraz bezpieczeństwo. Zapewnia także skalowalność rozwiązania (można praktycznie dowolnie rozbudować system, nie napotykając na problem maksymalnej wydajności pojedynczej maszyny).

*BOX* przekazuje zapytanie do wszystkich załadowanych do systemu modułów i jeśli któryś z modułów sam je obsługuje, wtedy generowana jest przez ten moduł odpowiedź, która jest następnie odsyłana klientowi. Po obsłużeniu zapytania przez wszystkie moduły i nie wygenerowaniu przez żaden z nich odpowiedzi, zapytanie przekazywane jest dalej do serwera WWW.

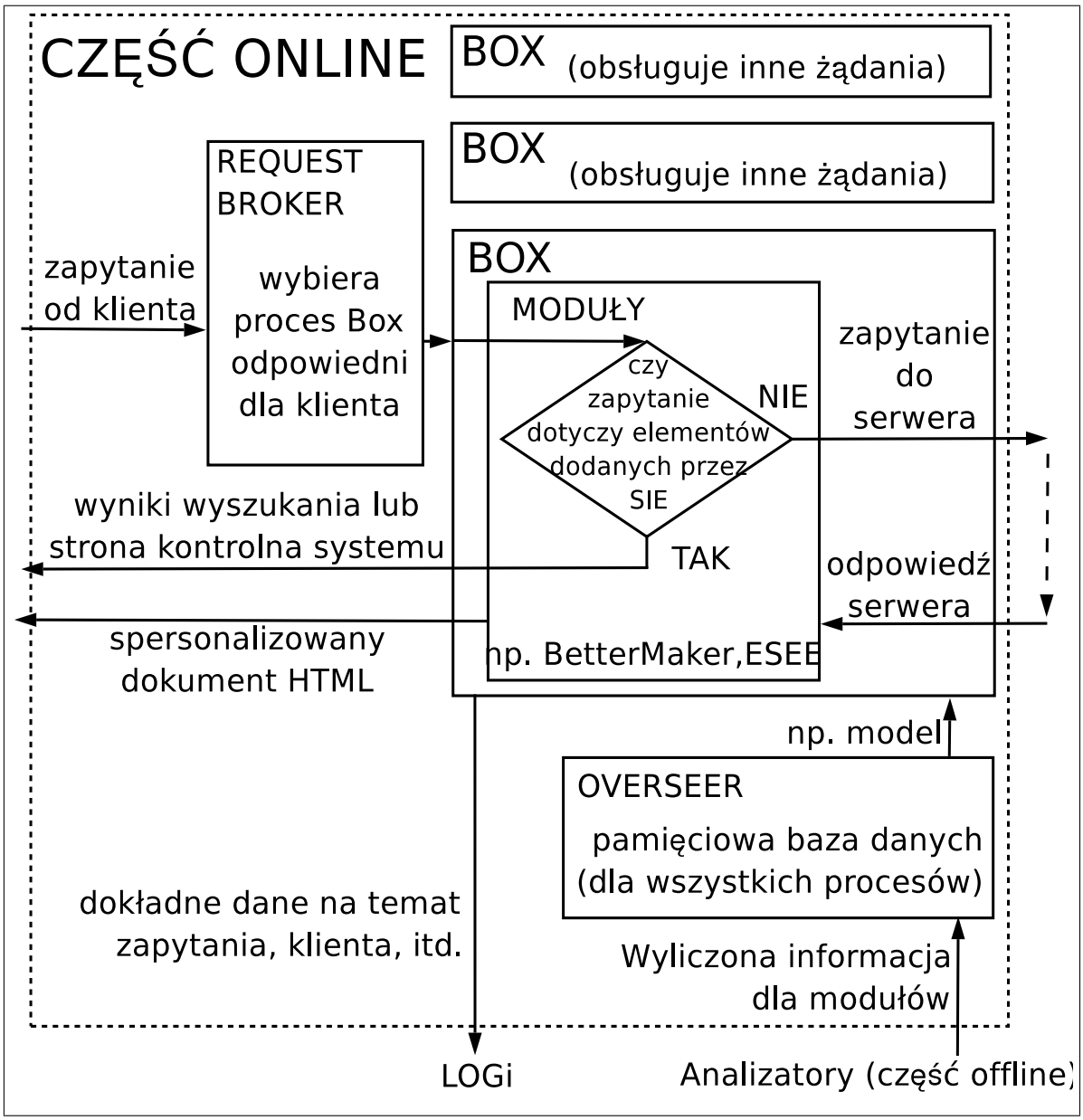
Następnie zapytanie trafia do serwera, który myśli, że to zapytanie pochodzi od naszego serwera i wysyła mu odpowiedź, nie zdając sobie sprawy, że jesteśmy jedynie pośrednikiem. (Uzyskujemy taką sytuację dzięki podmianie linków).

Powracające zapytanie znowu trafia do tego samego *BOKSA*, który tym razem przekazuje je do wszystkich modułów obsługujących powracającą odpowiedź.

Proces *BOX* modyfikuje także wszystkie linki znajdujące się na tej stronie tak, aby wskazywały na nasz serwer. Gdy klient kliknie na tak zmieniony link, jego przeglądarka połączy się z naszym serwerem, zamiast z serwerem docelowym. Mechanizm ten nazywamy *podmianą linków*. Dzięki niemu kontrolujemy komunikację pomiędzy klientem a serwerem WWW i możemy odnotowywać dodatkowe szczegółowe informacje, jak np. z jakiej strony pochodzi dane kliknięcie, czas czytania strony etc., które zapisujemy w bazie danych.




Rysunek 3.1: Schemat pokazujący ogólny widok całego systemu




Rysunek 3.2: Schemat pokazujący moduły składające się na *CZĘŚĆ ONLINE* systemu

Linki zewnętrzne (poza serwis) także są przepisywane, aby uzyskać w statystykach informację o wyjściach z obsługiwanego serwisu.

Powracających klientów identyfikujemy dzięki tzw. mechanizmowi ciasteczek [29]. Każdemu nowemu klientowi przydzielamy losowo wygenerowany identyfikator. Przeglądarka klienta pamięta od tej pory ciasteczko[1] i przy komunikacji z naszym serwerem przekazuje nam jego kopię, jednoznacznie identyfikując klienta.

Istotną częścią podsystemu *ONLINE* jest *WATCHDOG* — monitorowanie przez *REQUEST BROKERA* i automatyczne odłączanie nieprawidłowo działających (*BOKSÓW*) lub wyłączenia systemu jako całości i przepuszczania zapytań bezpośrednio do serwera WWW.

## 3.3. Część Offline

Na *CZĘŚĆ OFFLINE* składają się wszystkie operacje, które nie mają bezpośredniego związku z obsługą zapytań klientów. Są one wykonywane na innym komputerze lub w czasie mniejszego obciążenia systemu.

Przeprowadzana jest analiza danych zebranych w *CZĘŚĆI ONLINE*. Generuje ona wzorce, które zostaną następnie z powrotem przekazane do części online dla spersonalizowanego generowania treści na stronach serwisu.

*CZĘŚĆ OFFLINE* jest w małym stopniu implementowana przez system *SIE*, natomiast składają się na nią głównie programy analizujące, oferowane przez moduły. System *SIE* oferuje modułom interfejs programistyczny, który pozwala im na dostęp, do logów drzew sesji klientów. W skład sesji wchodzą także zapytania obsłużone przez moduły.

Dostępne są aktualnie 2 moduły, które powinny być uruchamiane regularnie[2], działające w *CZĘŚCI OFFLINE*:

- *EXPERIMENTER* — program dokonujący analizy dróg użytkowników. Wylicza informacje niezbędne do późniejszego stworzenia zindywidualizowanej struktury stron internetowych, zaspokajającej potrzeby każdego użytkownika. Szczegóły można znaleźć w dokumentacji [4]. Analizator ten współpracuje z modułem części online *BETTER MAKER*.

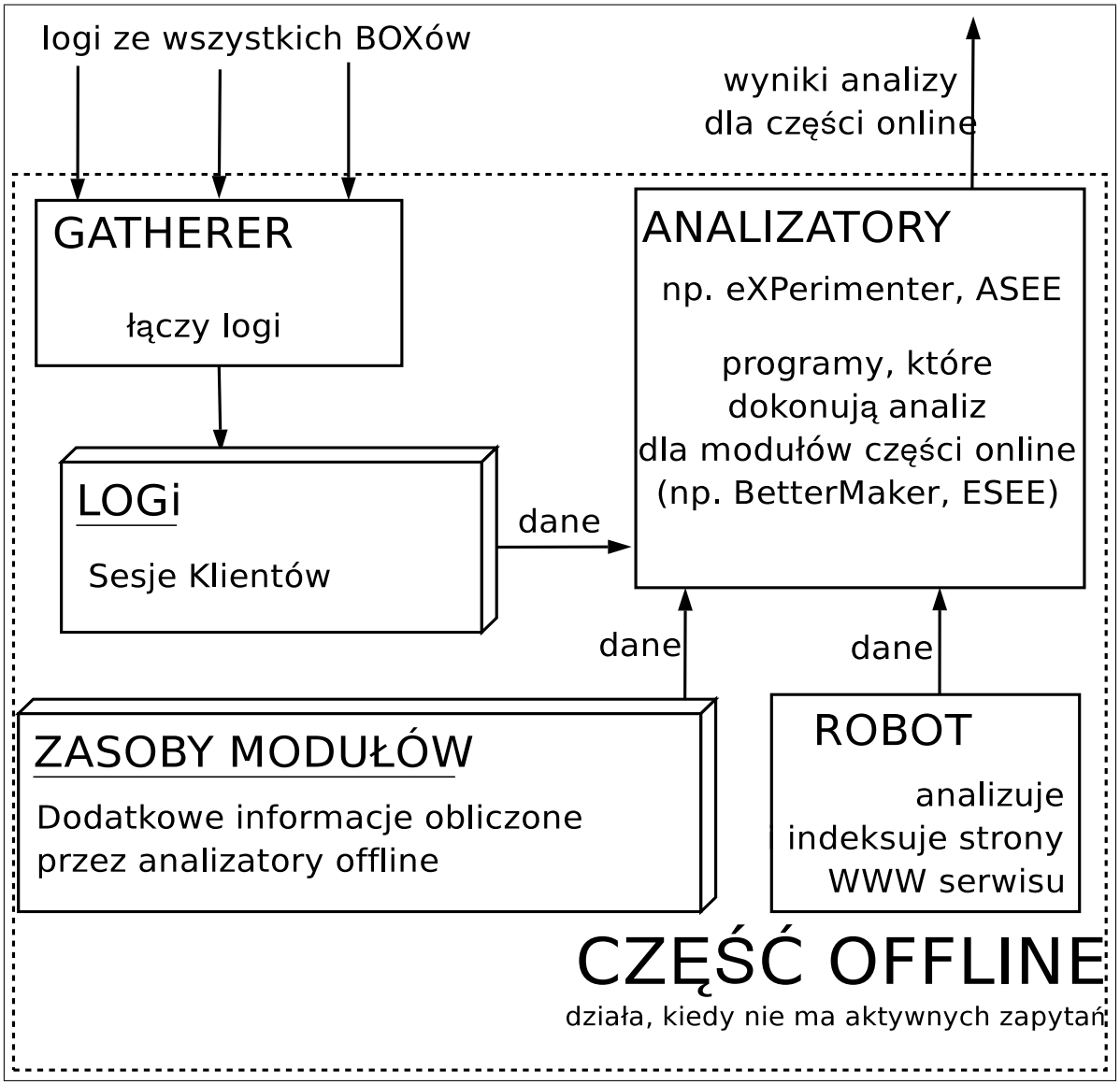
- *ASEE* — program analizuje zbiory stron na serwerze i ich ocenę przez użytkowników, aby uzyskać informacje umożliwiające sprawne działanie wyszukiwarki. Moduł ten współpracuje z *ESEE* — modułem części online. Dodatkowe informacje można znaleźć w dokumentacji tego modułu [3].

Ponadto do *CZĘŚĆI OFFLINE* można zaliczyć także *ODTWARZACZ SESJI*, który umożliwia odtwarzanie sesji nagranych przez system. Dzięki temu można testować działanie systemu w sytuacjach, gdy wielu użytkowników będzie próbowało wykonać pewne zestawy czynności (takie jak założenie konta w systemie). Odtwarzacz współpracuje z edytorem sesji. Główną zaletą mechanizmu nagrywania i odtwarzania sesji jest możliwość automatycznego wielokrotnego i równoległego (też z różnych komputerów jednocześnie) [3] symulowania działań użytkowników. Może to być wykorzystane np. do testowania wydajności serwera WWW. Poza tym bardzo duże możliwości daje parametryzowanie odtworzenia sesji (np. dla dowolnego użytkownika zapisujemy sesję będącą założeniem konta

[1] Niektóre starsze przeglądarki nie obsługują mechanizmu ciasteczek — w nowszych można ten mechanizm wyłączyć — wtedy system za pomocą mechanizmu *podmiany linków* potrafi rozpoznawać danego klienta przez czas trwania danej sesji, ale w następnej sesji klientowi zostanie przydzielony inny numer.

[2] na przykład za pomocą demona `cron`.

[3] Dzięki mechanizmowi tzw. *tuneli SSH* możemy „oszukiwać” serwer, który myśli, że zapytania przychodzą z wielu komputerów, a tak na prawdę pochodzą tylko z jednego.




Rysunek 3.3: Schemat pokazujący moduły składające się na *CZĘŚĆ OFFLINE* systemu

na nowym serwerze, a następnie odtwarzamy tę sesję wielokrotnie, podstawiając za każdym razem nazwę kolejnego użytkownika). Może to bardzo przyspieszyć procesy wielokrotnego wykonywania żmudnych, niewiele się od siebie różniących czynności.

## 3.4. Wątek akceptujący i wątki obsługujące

W poprzedniej architekturze *SIE* istniał jeden wątek akceptujący, natomiast dla każdego połączenia klienta tworzony był nowy wątek obsługujący.

Ponieważ nowy moduł komunikacyjny *SIE* napisany jest z użyciem asynchronicznego I/O, wszystkie operacje blokujące wszystkich wątków mogą być obsługiwane za pomocą tylko jednego wątku, który wchodzi w główną pętlę zapytywania o zdarzenia[4]. Dlatego tylko jeden wątek czeka na zdarzenia gotowości odczytu od klientów, jak i akceptuje nowe połączenia.

W przypadku, kiedy zapytanie klienta nie jest pełne, nie jest ono jeszcze obsługiwane. Kiedy całe zapytanie uda się sparsować, w tym momencie zakładamy że cały system może się zająć obsługą tego zapytania. Dzięki takiej architekturze martwe połączenia lub klienci wolno piszący nie są w stanie zatrzymać całego systemu.

## 3.5. Centralna baza oraz konfiguracja

Do systemu dołączona jest centralna baza danych *OVERSEER*. Posiada ona trzymane w pamięci tablice haszujące, które procesy *BOX* mogą odpytywać oraz zmieniać.

Do tego celu w interfejsie programistycznym systemu dostępne jest Api.Db, które dostarcza zarówno samemu *SIE*, jak i modułom następujących operacji:

- `init nazwa callback` — inicjalizacja dostępu do tablicy haszującej o podanej nazwie. Drugi parametr jest funkcją wołaną w *SIE* lub module, kiedy zawartość tabeli zmieni się w bazie danych. Jest ona wymagana w przypadku modeli wyliczanych przez *CZĘŚĆ OFFLINE*, które po wywołaniu tej funkcji mogą zostać ponownie odczytane przez wszystkie procesy *BOX*.
- `get klucz` — pobranie wartości związanej z danym kluczem.
- `set klucz wartość` — ustawienie podanej wartości.

Ponadto dostępna jest nazwa tablicy używanej przez system do przechowywania konfiguracji. Podczas inicjalizacji centralna baza wczytuje plik konfiguracyjny, a następnie przekazuje wszystkie parametry podłączającym się do niej procesom *BOX*.

Tablice centralnej bazy są trzymane tylko w pamięci. Podczas jej uruchamiania wymagane tablice z modelami na przykład dla wyszukiwarki *ESEE* lub modułu *BETTER MAKER* można załadować do *OVERSEERA* za pomocą procesów *PUT*.

## 3.6. Request Broker oraz WatchDog

Dodatkowo niezbędnym elementem systemu jest proces *REQUEST BROKER* dzielący nadchodzące żądania pomiędzy procesy przetwarzające, gdyż mogą się one znajdować na różnych elementach klastra.

Proces ten jest jedynym procesem, z którym kontakt ma bezpośrednio klient. Odbiera on wszystkie żądania, nie dokonuje pełnego parsowania, a jedynie bardzo pobieżnie próbuje rozpoznać Cookie

[4] W niektórych mechanizmach jest to wymagane, na przykład `epoll_wait`.

i adres docelowy linku. Na podstawie tych informacji przekazuje całość komunikacji odpowiedniemu procesowi *BOX*.

Jest on tak napisany, aby cała komunikacja, odbywająca się w danej sesji między klientem a serwerem, trafiała do tego samego procesu przetwarzającego (jeśli jest to możliwe). Dzięki temu można potencjalnie zmniejszyć ilość wymaganej komunikacji z centralną bazą danych. Komunikacja z bazą danych będzie miała miejsce, kiedy proces *BOX* obsługujący klienta zostanie wyłączony (lub nie działa) i dalsze zapytania będą kierowane do innego procesu, który nie będzie posiadał w pamięci podręcznej informacji o kliencie.

Podczas pisania *SIE* dołożono wszelkich starań, aby uniknąć błędów, ale nie jest to całkowite możliwe, w szczególności moduły załadowane do systemu mogą powodować błędy lub się pętlić (tak jak zostało opisane w rozdziale 2.2.6).

Dlatego w proces *REQUEST BROKER* wbudowany został mechanizm *WATCHDOG*, który po wysłaniu każdego zapytania od klienta do procesu przetwarzającego, czeka odpowiednią ilość czasu (domyślnie 3 sekundy). Jeśli w tym czasie od systemu nie nadejdzie odpowiedź na zapytanie, proces *BOX* obsługujący zapytanie traktowany jest jako martwy, a zapytanie jest kierowane do prawdziwego serwera WWW.

Motywacja dla przekierowania zapytania do serwera, a nie do kolejnego procesu *BOX* jest następująca:

- Wszystkie procesy *BOX* są identyczne i jeśli jeden nie był w stanie odpowiedzieć na zapytanie, ze względu na błąd implementacji, pozostałe też nie będą w stanie. Przekierowywanie zapytania do *BOKSÓW* powodowałoby wyłączenie całego systemu.

- Jeśli aktualnie obciążenie zapytaniami jest na tyle duże, że proces *BOX* nie jest w stanie odpowiedzieć, pozostałe też nie będą w stanie, natomiast przekierowanie zapytań bezpośrednio do serwera pozwoli je zmniejszyć.

# Rozdział 4

# Rzeczywiste zastosowanie systemu

## 4.1. Możliwości zastosowania systemu

System *SIE* został zaprojektowany, aby stał przed dużym serwisem WWW i optymalizował oraz personalizował jego strukturę. Dzięki możliwościom podpowiadania linków, udostępnieniu wyszukiwarki, możliwości nagrywania i późniejszego odtwarzania sesji, system *SIE* upraszcza pisanie serwisu WWW.

Dzięki nowemu modułowi, można uruchomić *SIE* jako pośrednik przed dużym serwisem, gdzie szybkość przetwarzania oraz bezpieczeństwo (dostępność) są istotne. Ponadto dzięki oddzieleniu serwera WWW systemu, możliwe jest wykonywanie części operacji wymaganych do działania serwisu w samym *SIE*, takich jak liczenie sesji czy logowanie.

Podstawową funkcjonalnością *SIE* jest jednak udostępnianie interfesju programistycznego, pozwalającego osobie chętniej zaimplementować dowolne rozwiązanie z dziedziny „samoadaptującej się sieci” na zrobienie tego, bez przejmowania się takimi szczegółami jak parsowanie HTTP i HTML czy pamiętanie o sesjach użytkowników. Ponadto *SIE* umożliwia łatwe rozdzielenie złożonych procesów obliczeniowych na różne elementy klastra. Możliwe jest też oddzielenie części obliczeniowej, która nie jest związana z bezpośrednim odpowiadaniem na żądania klienta i wykonywanie jej *OFFLINE*, a następnie wygodne przekazywanie informacji o nowych modelach *CZĘŚĆI ONLINE*.

## 4.2. Przykłady zastosowania

W systemie aktualnie działają dwa moduły:

- **dynamiczne dodawanie spersonalizowanej struktury linków** — moduł *BETTER MAKER* dodaje do każdej strony strukturę linków, skonstruowaną w oparciu o analizę działań użytkowników,

- **udostępnienie spersonalizowanej wyszukiwarki** — wyszukiwarka jest dostępna z poziomu panelu kontrolnego na każdej stronie (moduł *SEE*) — jej działanie jest także oparte na analizie działań użytkowników.

Ponadto *SIE* pozwala także na zapisanie sesji oraz na jej późniejsze odtwarzanie, także w wersji sparametryzowanej (język *WTL* [5]). Nie jest to zaimplementowane jako moduł online, a jedynie program korzystający *OFFLINE* z informacji o sesjach zawartych w normalnych logach *SIE*.

## 4.3. Pierwsze wdrożenie

Pierwsza wersja systemu została uruchomiona w połowie marca 2003 na serwisie olimpiady informatycznej (sio.mimuw.edu.pl:8080). Od początku kwietnia 2003 system oferował już swoją wyszukiwarkę i podpowiadał linki. Miał jednak tyle niedociągnięć, że nie nadawał się do sensownego wykorzystania.

W ciągu trzech miesięcy działania zostało zarejestrowane 113 sesji, w których brało udział 86 użytkowników. W sumie klienci zadali 473 zapytania HTTP[1], z tego 63 do naszej wyszukiwarki.

## 4.4. Aktualne wdrożenie

System z nowym modułem komunikacyjnym uruchomiony był na serwisie pracowniczym MIMUW (duch.mimuw.edu.pl:8079), gdzie był testowany jedynie przez niedużą grupę osób rozwijających dalej *SIE*.

Zostało zarejestrowane 39 sesji (nie licząc sesji testowych i robotów), jednak ponieważ było to wdrożenie testowe, a sam serwis nie oferuje istotnych stron, dlatego dane te są głównie potwierdzeniem stabilności systemu.

## 4.5. Wydajność

Wyniki wszystkich testów zostały wyliczone na komputerze:

- sprzęt: Centrino 1.8, 1Gb RAM, 2048kb L2 Cache,
- system: Debian GNU/Linux 3.1 (SID),
- jądro: Linux 2.6.11.5.

Ustawione zostały następujące zmienne konfiguracyjne:

- /proc/sys/fs/file-max = 131072
- /proc/sys/net/ipv4/tcp_fin_timeout = 15
- /proc/sys/net/ipv4/tcp_max_syn_backlog = 16384
- /proc/sys/net/ipv4/tcp_tw_reuse = 1
- /proc/sys/net/ipv4/tcp_tw_recycle = 1
- /proc/sys/net/ipv4/ip_local_port_range = 1024 65535

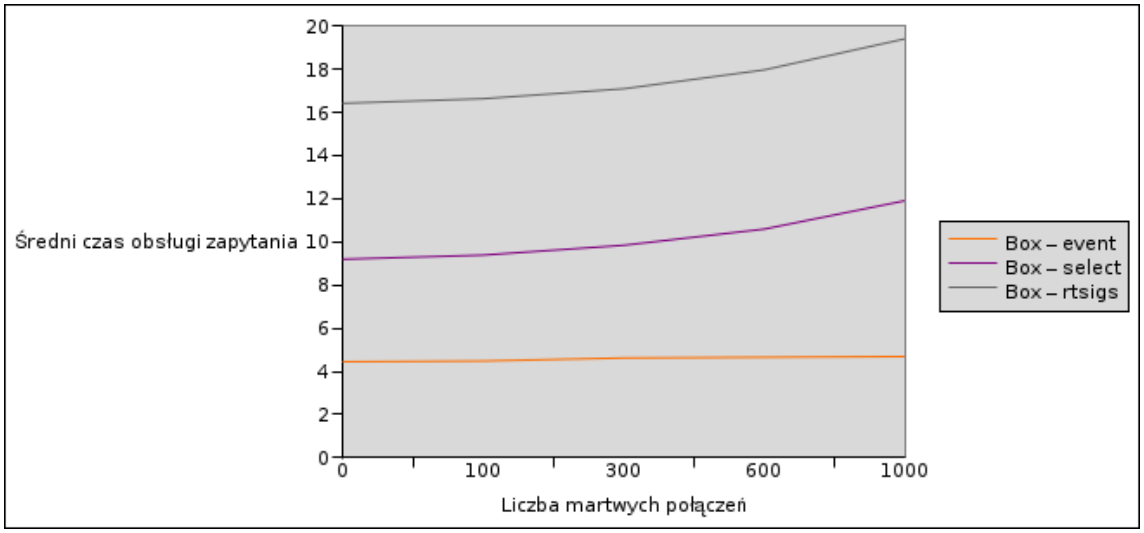
Jako serwer WWW został wykorzystany Apache w wersji 2.0.53.

Pierwszym z przeprowadzonych testów jest sprawdzenie, jak działa nowy moduł systemu *SIE* w zależności od używanego modułu komunikacji. Do stworzenia martwych połączeń został użyty program `deadconn` opisany we wstępie. Porównane zostały:

- `Box-select` — wersja systemu korzystająca z modułu Unix biblioteki standardowej OCamla, która wewnętrznie wykorzystuje wywołanie systemowe `select`.

[1] Nie licząc robotów, patrz dodatek B.

- `Box-event` — wersja korzystająca z ocamlowej warstwy pośredniej dookoła biblioteki libevent, korzystająca z mechanizmu `EPoll`.
- `Box-rtsigs` — wersja korzystająca z biblioteki libevent, po ustawieniu zmiennej środowiskowej `EVENT_NOEPOLL`.




Rysunek 4.1: Wykres pokazujący zależność średniego czasu obsługi zapytania w milisekundach od liczby martwych połączeń utworzonych za pomocą `deadconn`

Test ten symuluje dużą liczbę połączeń martwych lub wolnych, z którymi często styka się serwer o dużym obciążeniu, jak również może symulować atak typy DoS na serwer za pomocą mechanizmu tego rodzaju, mający na celu uniedostępnienie serwera.

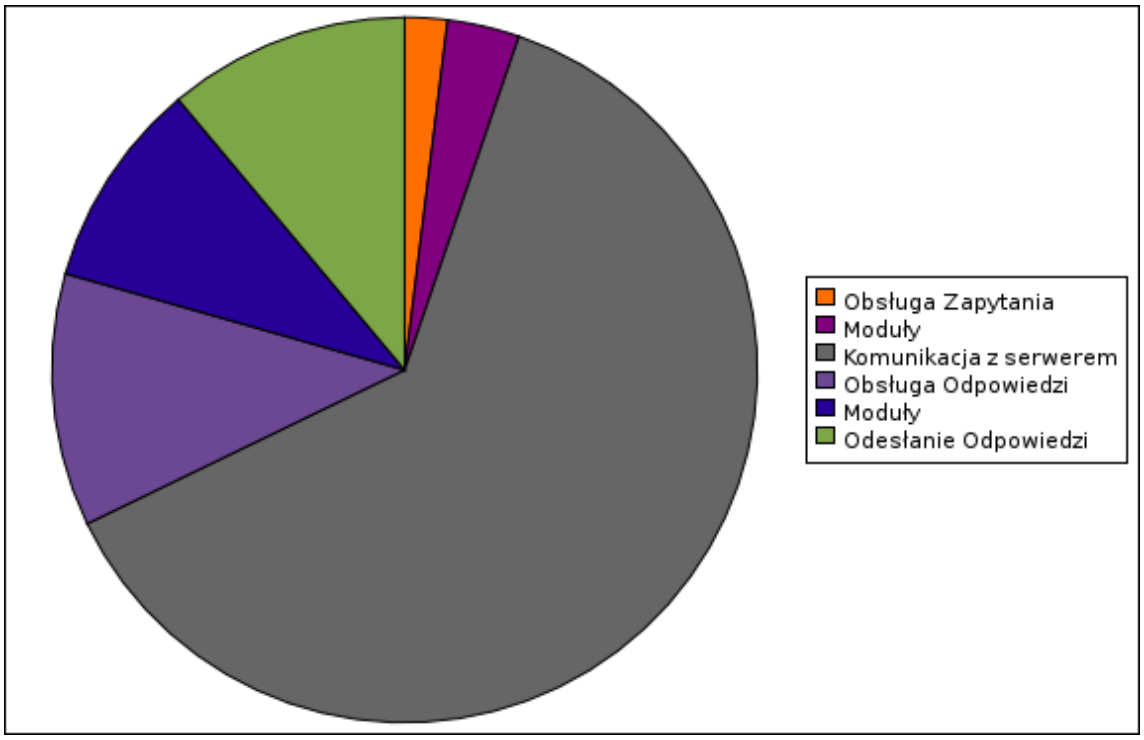
Na wykresie 4.1 widać że dla mechanizmu `EPoll` (`Box-event`), czas potrzebny na obsługę żądań nie rośnie proporcjonalnie do liczby połączeń, przy mechanizmie `select` (`Box-select`) zależność jest liniowa. Porównany został także mechanizm `Real-time signals` (`Box-rtsig`), w którym dostarczenie informacji o ukończonej operacji wejścia/wyjścia następuje, poprzez dostarczenie sygnału. Nie wymaga ono żadnych nakładek na standartowe jądro w wersji 2.4. Jednak choć teoretycznie rozwiązanie powinno dawać skalowalność lepszą niż liniowa, wydajność nie jest zadowalająca, gdyż mechanizm wymaga ustawienia oczekiwania na sygnał (`sigwaitinfo`) wewnątz normalnej pętli `poll`.

Jednocześnie przeprowadziłem taki sam test, mierząc wydajność samego serwera WWW[2] i jego wydajność wynosi średnio 2.406 milisekund na zapytanie, czyli jest on w stanie odpowiadać około 1.849 raza szybciej niż system *SIE* z niego korzystający.

Kolejnym testem jest mierzenie wydajności wewnętrznej — czyli części systemu *SIE* składających się na obsługę pojedynczego zapytania. Do zmierzenia tych wartości zostały dodane w programie linie mierzące czas i program jest wywoływany z pewną liczbą zapytań nierównoległych, między którymi czas jest odejmowany. Należy zauważyć, że czas oznaczony jako “Komunikacja z Serwerem” zawiera zarówno czas potrzebny na wysłanie zapytania i odebranie odpowiedzi (ale bez ich parsowania) jak i czas oczekiwania na sam serwer.

Jak widać na wykresie 4.2 czas potrzebny na obsługę zapytania przez serwer stanowi główną część obsługi zapytań. Drugim najbardziej czasochłonnym miejscem, jest obsługa zapytania i od-

[2]Przy czym testowana była wydajność jedynie bez martwych połączeń, gdyż mechanizmy obsługi wolnych połączeń wbudowane w serwer Apache są istotnie innego rodzaju.




Rysunek 4.2: Wykres pokazujący wydajność wewnętrzną systemu *SIE*.

powiedzi przez moduły działające w systemie. Całe parsowanie zarówno HTTP jak i HTML oraz modyfikacja stron są istotnie krótsze.

Do zmierzenia czasów przedstawionych na tym wykresie w różnych miejscach programu zostały dołożone sprawdzania czasu i mierzenie odstępów pomiędzy tymi momentami. W tym teście zapytania były zadawane tylko sekwencyjnie aby wyniki nie kolidowały

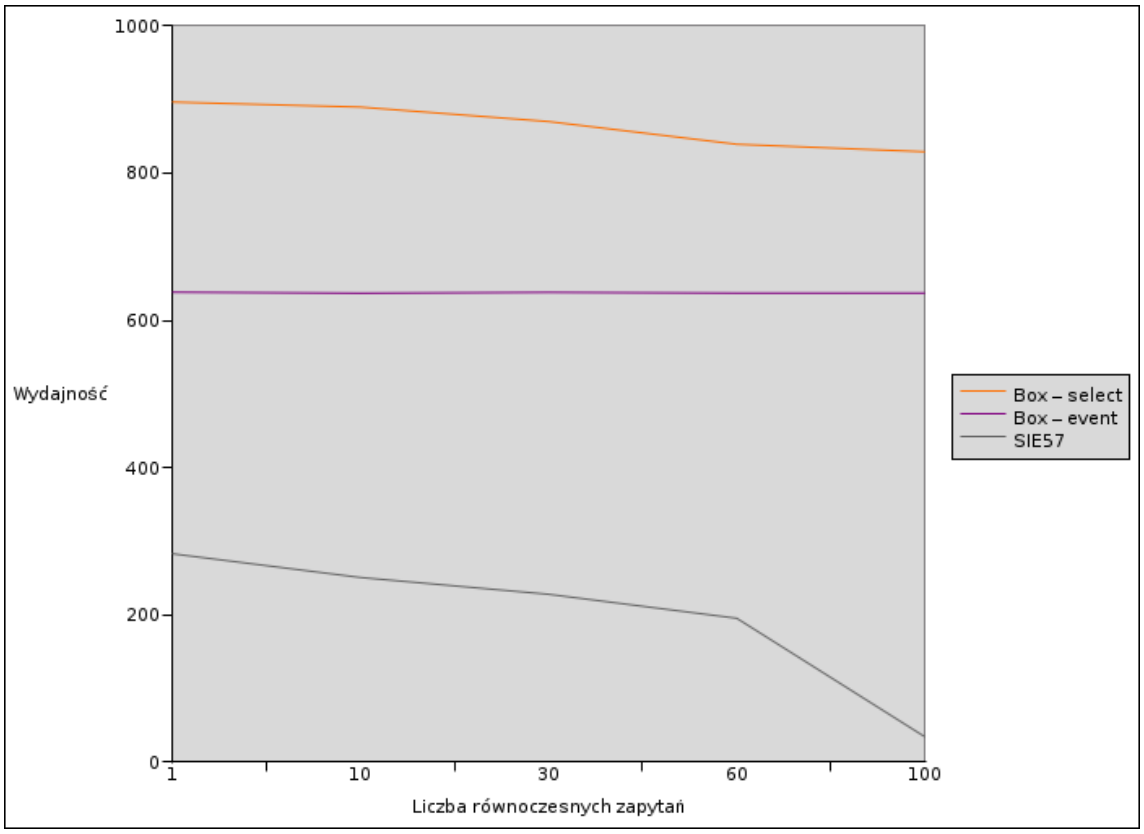
## 4.6. Porównanie z poprzednią wersją

Na tej samej konfiguracji przeprowadzony został także test porównujący wydajność *SIE* w zależności od liczby połączeń równoległych. Test ten przeprowadzony został także dla wersji z poprzednim modułem komunikacyjnym.

Test ten różni się tym, od poprzedniego, że nie ma martwych połączeń, wszystkie połączenia do serwisu (serwera) wysyłają kompletne zapytanie i po odebraniu odpowiedzi się kończą. Test ten pokazuje wydajność serwera w sytuacjach normalnego obciążenia — to znaczy pewnej liczby użytkowników jednocześnie zapytujących o strony serwisu.

Wyniki testu mierzącego zależność wydajności od liczby rółnoczesnych zapytań (wykres 4.3) pokazują, że wydajność *SIE* korzystającego z mechanizmu `select` (`Box-select`) spada wraz ze wzrostem liczby równolegle zadawanych zapytań. Jednocześnie widać, że w przypadku jednego wątku korzystającego z mechanizmu `EPoll` (`Box-event`), wydajność prawie się nie zmniejsza.

Jednocześnie widać, że wydajność poprzedniego modułu komunikacyjnego *SIE* (`SIE57`) jest istotnie niższa. Poprzedni moduł komunikacyjny dla każdego nowego połączenia tworzy nowy wątek, który je obsługuje za pomocą mechanizmu `select`. Duża liczba wątków powoduje duże obciążenie zarówno procesora jak i pamięci serwera, na którym uruchomione jest *SIE*. Wartość na wykresie dla 100 połączeń równoległych jest mniejsza niż powinna być, gdyż test dla tak dużej liczby połączeń się nie powiódł, 12% połączeń zwróciło niepoprawne odpowiedzi HTTP i nie zostały one wzięte pod




Rysunek 4.3: Wykres pokazujący zależność wydajności (w kilobajtach na sekundę) od liczby równoczesnych zapytań.

uwagę w wartości.

## 4.7. Podobne systemy

Jedną z istotnych zalet systemu *SIE* jest brak innego oprogramowania oferującego podobną funkcjonalność. Istnieją programy oferujące funkcjonalność pojedynczych modułów, jednak nie istnieją programy oferujące podstawowej funkcjonalności *SIE* — czyli oferowania interfejsu programistycznego modułom użytkownika, pozwalając na wygodną obsługę protokołów sieciowych, parsowanie i dostęp do drzewa HTML, rozpoznawanie użytkowników oraz ich działań oraz przechowywanie tych informacji w bazie danych.

Przegląd implementacji systemów z dziedziny Web Usage Mining został przedstawiony w pracy doktorskiej Roberta Cooley'a [14]. Została tam zamieszczona klasyfikacja omawianych systemów na pięć kategorii:

1. *personalizacja*,
2. *usprawnienie systemu*,
3. *modyfikacja stron*,
4. *inteligencja biznesowa*,
5. *charakteryzacja użycia*.

Aktualne moduły w *SIE* (*ADAPTER* i *SEE*) można zaliczyć do dziedzin *personalizacja* oraz *modyfikacja stron*. Dalsze moduły pisane dla systemu mogą być zaliczane do dowolnej z powyższych kategorii.

IBM stworzył własny zestaw narzędzi do tworzenia pośredników HTTP, WBI — Web Browser Intermediaries[3]. Aktualnie WBI jest częścią WebSphere Transcoding Publisher i zestaw narzędzi programistycznych (Development Kit) nie jest dostępny do ściągnięcia. WBI jest opisane w [10], wprowadzona jest tam idea „pośredników" jako elementów obliczeniowych na ścieżce komunikacji sieciowej, a WBI jest zestawem narzędzi do budowania i uruchamiania modułów pośredniczących. Jak jest opisane w [11], WBI jest umieszczone między jednym użytkownikiem, a Internetem (wszystkimi serwerami), podczas gdy *SIE* analizuje informacje odwiedzin wszystkich użytkowników na jednym serwisie.

Wartymi wspomnienia programami, które mają funkcjonalność modułów *SIE*, są:

- WebCANVAS [12] — system analizujący logi serwera i wyświetlający graficznie wzory poruszania się po stronach WWW serwisu. Jest to osiągnięte dzięki automatycznemu klastrowaniu użytkowników oraz ręcznemu podzieleniu stron serwisu na zestaw kategorii.
- IndexFinder [18] — system który tworzy „strony indeksowe" serwisu. Są to strony, które zawierają linki do podobnych stron. Grupowane są zarówno strony odwiedzane przez tych samych użytkowników, jak i strony mające podobną zawartość. Administrator strony może następnie wybrać, które ze stron indeksowych chce dołączyć do serwisu.
- Systemy Proteus i Montage (opisane w pracy doktorskiej Corina Andersona [9]. Pierwszy z nich zmienia strony WWW tak, aby można było je łatwiej wyświetlać na urządzeniach o małych ekranach, takich jak PDA lub telefony komórkowe. Drugi buduje portal internetowy dla użytkownika, na podstawie linków i zawartości stron przez niego wcześniej odwiedzanych.

[3] Wcześniej Web Browser Intelligence.

# Rozdział 5

# Podsumowanie

Przedstawiony został nowy, działający moduł komunikacyjny dla systemu *SIE*. Umożliwia on uruchomienie tego systemu dla serwisów, gdzie szybkość działania i niezawodność są istotne. Uproszczony został proces kompilacji i instalacji systemu oraz dodana została centralna konfiguracja.

W nowym module komunikacyjnym użycie asynchronicznego wejścia-wyjścia w połączeniu z klastrową architekturą *SIE* umożliwia jego praktycznie dowolne skalowanie. Wspólny interfejs dla modułów pozwala na ich dowolne dodawanie do systemu, a przekazywanie konfiguracji poprzez centralną bazę danych umożliwia prostą synchronizację stanu zmiennych w systemie.

Według mnie system *SIE* wraz z nowym modułem komunikacyjnym jest udanym projektem, mającym na celu wykorzystanie komputera oraz nowych technik dzięki niemu dostępnych, do zmniejszenia nakładów czasowych i kosztowych wymaganych zarówno do stworzenia serwisu WWW, jak i jego personalizacji i optymalizacji dla użytkowników. Zmniejsza on także koszty samego użytkownika w znajdowaniu informacji.

Mam nadzieję, że powstaną profesjonalne moduły do systemu *SIE*, mogące wykorzystać całą moc jego przetwarzania i że system wraz z nowym modułem przyczyni się do ułatwienia użytkownikom poruszania się po ogromie informacji dostępnych w internecie.

## 5.1. Dalsze możliwości rozwoju

Istotną cechą, której brakuje dotychczas systemowi *SIE*, jest brak obsługi HTTPS [25, 26, 27], czyli połączeń szyfrowanych za pomocą SSL. Pozwala to na szyfrowanie komunikacji pomiędzy użytkownikiem a serwerem, tak aby nie było możliwe odczytanie przesyłanych danych przez osoby trzecie pośredniczące w przekazywaniu pakietów.

Istniejące moduły obsługują jedynie zawartość tekstu stron, jednak możliwe jest także wykorzystanie *SIE* do innych modyfikacji zawartości. Przykładem tego może być skalowanie obrazów na stronach WWW do mniejszych rozmiarów, co może być bardzo wygodne przy oglądaniu stron za pomocą komputerów przenośnych lub telefonów komórkowych o małym obszarze wyświetlania.

Ponadto *SIE* można łatwo wykorzystać do tworzenia stron dostosowanych do potrzeb osób starszych lub niedowidzących. Podobne systemy adaptujące i sortujące strony w zależności od ich zawartości są opisane na przykład w [50], jednak ich implementacja wymagała dużego nakładu na zaimplementowanie technik i narzędzi już dostępnych w *SIE*.

Możliwe jest także generowanie całej zawartości stron przez moduły, dzięki czemu możliwe jest w bardzo prosty sposób dodanie serwowania stron znajdujących się fizycznie jako pliki na komputerze, na którym jest zainstalowane *SIE*[1]. Taka funkcjonalność dodatkowo zwiększyłaby wydajność

[1] W tej chwili będący integralną częścią *SIE* moduł Panel nie tylko dodaje panelik będący na każdej stronie, lecz także dodaje stronę kontrolną systemu.

*SIE*, jednak uniemożliwiłaby ona działanie *SIE* dla zbiorów serwisów działających na różnych komputerach. W szczególności aktualnie możliwe jest uruchomienie prywatnie *SIE* dla całego internetu i wykorzystywanie funkcjonalności niezależnej od serwisu, takiej jak nagrywanie sesji.

Ponadto dzięki możliwości generowania stron bezpośrednio przez moduły, możliwe jest w bardzo prosty sposób dodanie funkcjonalności podobnej do PHP czy CGI, czyli obsługi stron napisanych w języku funkcyjnym, które są doprowadzane do postaci HTML dopiero przez *SIE*.

# Dodatek A

# Podręcznik administratora

## A.1. Wymagania

Do kompilacji systemu *SIE* wymagany jest system kompatybilny z Unixem. Aktualna wersja była testowana pod `Linuxem` oraz systemem `MS Windows` z pomocą zestawu narzędzi `Cygwin`.

System powinien dać się uruchomić pod dowolnym systemem z podstawowymi narzędziami uniksowymi. W szczególności potrzebne są do tego:

- Narzędzia powłoki wymagane do działania skryptu `configure` czyli shell oraz podstawowe narzędzia używane przez skrypty shella, takie jak: sed, grep, itp.
- Kompilator `gcc` oraz system `make`.
- Kompilator OCamla, w wersji co najmniej 3.04. System *SIE* nie nakłada wymagań na wersję kompilatora OCamla, lecz używane przez niego biblioteki wymagają obsługi strumieni za pomocą składni wbudowanej. System był testowany pod kompilatorami 3.06, 3.07+2 oraz 3.08.3.
- Opcjonalnie *SIE* może korzystać z biblioteki libpcre do rozpoznawania rozszerzonych wyrażeń regularnych. Jeśli takowa nie jest zainstalowana, zostanie użyta wersja biblioteki dołączona do *SIE*.
- Opcjonalnie *SIE* może korzystać z biblioteki libevent zainstalowanej w systemie. *SIE* może także korzystać z wersji biblioteki dołączonej do niego lub można całkowicie wyłączyć korzystanie z biblioteki libevent i korzystać z dodatkowej wersji ogólnych komend obsługi synchronicznego wejścia-wyjścia napisanej z pomocą modułu `Unix` biblioteki standardowej OCamla.

## A.2. Kompilacja

Kompilację rozpoczynamy za pomocą uruchomienia skryptu:

```
./configure
```

Skrypt ten próbuje odnaleźć wszystkie parametry wartości zmiennych zależnych od systemu używanych podczas kompilacji. Używa ich do stworzenia pliku `Config.mk` w katalogu głównym, który następnie będzie używany w procesie kompilacji.

Plik `configure` został utworzony z pliku `configure.in` i w przypadku gdyby zaistniała potrzeba jego zmiany, należy zmodyfikować plik wejściowy i ponownie uruchomić programy `aclocal` i `autoconf`. Więcej szczegółów w dokumentacji tych programów.

Następnie do kompilacji całego systemu należy wydać polecenie:

```
make
```

Program ten, najpierw wyliczy zależności między plikami, a następnie skompiluje cały system.

Podczas kompilacji systemu mogą wystąpić ostrzeżenia (ang. *warning*) o delikatnych dopasowaniach do wzorcach. Są one jednej z następujących postaci:

- `Warning: Bad style, (...),`
- `Warning: this pattern is unused,`
- `Warning: this pattern-matching is not exhaustive.`

Występują one tylko przy pewnych wersjach kompilatora OCaml. Jeśli występują, to tylko w bibliotekach używanych przez system (np. OCamlNet). Nie są one szkodliwe.

Po bezbłędnej kompilacji systemu, zostają stworzone wszystkie pliki binarne wymagane do działania systemu, zarówno dla części online jak i dla *CZĘŚCI OFFLINE*.

## A.3. Konfiguracja

W głównym katalogu systemu jest plik `sie.config`, w którym niezbędnie należy zmienić parametry, tak aby odzwierciedlały konfigurację serwisu WWW oraz ustawić parametry dla systemu *SIE*:

- SIE_BASEURL — adres oraz port, na którym *SIE* jest zainstalowane i na który mają być przepisywane linki.
- SIE_TARGETURL — adres do strony rzeczywistego serwera WWW.
- SERVER_PORT — port, na którym działa serwer WWW.
- SERVER_ADDR — adres, na którym działa serwer WWW.
- HOST — adres oraz port, na którym działa wyszukiwarka rzeczywistego serwera WWW, jeśli użytkownik zdecyduje się nie korzystać z SEE, lecz z wyszukiwarki wbudowanej w serwer.

Zmiany pozostałych parametrów, nie są wymagane do uruchomienia systemu, mogą być jedynie wykorzystywane do ulepszenia jego działania.

Większość parametrów ma nazwy odzwierciedlające ich znacznie, pozostałe są opisane w tym pliku konfiguracyjnym.

## A.4. Dostępne parametry linii poleceń

Centralna baza danych *OVERSEER* posiada następujące opcjonalne parametry dostępne z linii poleceń:

- `-iface ip1, ip2, ...` — lista interfejsów sieciowych, na których program nasłuchuje.
- `-port` — numer portu, na którym program nasłuchuje.

Główny proces przetwarzający *BOX* posiada następujące opcjonalne parametry, dostępne z linii poleceń:

- `-config plik` — nazwa pliku innego niż domyślny plik `sie.config`, jeśli dany proces *BOX* ma działać z inną konfiguracją niż otrzymana z centralnej bazy.
- `-overseer host:port` — adres, na którym *BOX* ma szukać centralnej bazy danych.

## A.5. Uruchomienie części online systemu

Uruchomienie systemu, tak aby zaczął zbierać logi i udostępniał panelik z sesją, wymaga uruchomienia następujących trzech procesów:

- `Overseer/Overseer`,
- `RequestBroker/RequestBroker`,
- `Box/Box`.

System zaczyna być widoczny na porcie takim jak w pliku konfiguracyjnym i zbiera informacje o sesjach użytkowników w pliku `basic.log`.

## A.6. Wyliczanie statystyk i aktualizacja w części online

Nowy moduł komunikacyjny nie dotyczy *CZĘŚĆI OFFLINE* systemu, jej uruchamianie jest identyczne jak wcześniej.

Główny engine systemu nie posiada żadnego procesu w *CZĘŚCI OFFLINE*, jedynie udostępnia biblioteki umożliwiające procesom offline modułów iterację po logach sesji klientów.

W skrócie na uruchomienie *CZĘŚCI OFFLINE* składa się:

- Ściągnięcie struktury serwisu za pomocą *ROBOTA* i wyliczenie parametrów stron za pomocą *ASEE*. Więcej informacji dotyczącej tego procesu w [3].
- Wyliczenie statystyk przechodzenia stron i wyliczenie modelu za pomocą procesu *XP*. Więcej informacji dotyczącej tego procesu w [4].
- Załadowanie modeli do centralnej bazy danych za pomocą procesów *PUT* opisanych w powyższych dokumentacjach.
- Wysłanie sygnału `SIGUSR1` do procesu *OVERSEER*, aby przeładował konfigurację i poinformował wszystkie procesy *BOX*, o zmianie modelu.

W przypadku, kiedy chcemy powyższy proces wykonywać automatycznie, na przykład co noc, należy powyższe czynności ustawić na przykład w skrypcie demona `cron`.

# Dodatek B

# Słownik

**Administrator** - Osoba nadzorująca działanie systemu.

**API** - Zestaw funkcji udostępnianych programiście przez oprogramowanie (ang. *Application Programming Interface*).

**Box** - Jeden z procesów przetwarzających żądania klienta. Na każdym z komputerów wchodzących w skład klastra przetwarzającego działa jeden program Box. Wewnętrznie posiada on ilość wątków i procesów odpowiednią dla optymalnego obsługiwania aktualnego poziomu częstości zapytań.

**Część offline systemu** - Fragment systemu nie związany bezpośrednio z żądaniami klientów. Jego wydajność nie jest krytyczna.

**Część online systemu** - Fragment bezpośrednio obsługujący konkretne żądania klientów serwisu *WWW*, na którym operuje system SIE. Działa w czasie rzeczywistym tylko wtedy, gdy klient serwisu prosi o jakiś zasób. Jej wydajność jest bardzo istotna, ma pierwszeństwo (przed częścią offline) w korzystaniu z zasobów systemu operacyjnego.

**GPL (General Public License)** - Licencja pozwalająca na nieodpłatne wykorzystywanie i modyfikację oprogramowania.

**GUI** - Graficzny Interfejs Użytkownika (ang. *Graphical User Inteface*).

**Klient** - Osoba podłączająca się do serwisu *WWW* obsługiwanego przez system *SIE* poprzez sieć. Klient zazwyczaj korzysta w tym celu z przeglądarki WWW. Klient chce otrzymać pewne informacje z serwerów, do których system jest podłączony lub skorzystać z usługi oferowanej przez system.

**Kryterium** - Funkcja służąca do oceniania atrakcyjności strony ze względu na pewną cechę. Wartości kryteriów stron należących do serwisu są wyliczane w części offline, aby być później wykorzystywanymi online do oceniania atrakcyjności strony dla konkretnego użytkownika w zależności od jego preferencji ([3],[4]).

**Log** - Informacja o zdarzeniach w systemie udostępniana przez niego modułom części offline.

**Moduł** - Część programu dająca się zaimplementować niezależnie od innych części. Moduły składające się na jeden program mogą być kompilowane wspólnie do tego programu. Moduł może być wykorzystywany przez kilka różnych programów.

**OverSeer** - Centralna baza danych udostępniająca zbiór hasz-tablic z możliwością modyfikacji i cacheowania. Pozwala na trzymanie danych współdzielonych przez różne procesy przetwarzające.

**Panelik** - Interfejs dla klientów serwisu. Pozwala im konfigurować system SIE (do pewnego stopnia). Jest doklejany do stron serwowanych przez system.

**Wtyczka** - Moduł do części online. Zadaniem wtyczek jest modyfikowanie zapytań klientów i odpowiedzi serwerów. Wtyczki korzystają z interfejsu programistycznego, aby nie musieć się zajmować się niskopoziomowymi zadaniami takimi jak wielozadaniowość i protokoły komunikacyjne.

**Request Broker** - Proces rozdzielający połączenia klientów pomiędzy procesy Box działające na różnych elementach klastra.

**Robot** - Program działający w sieci, który zbiera w sposób zautomatyzowany informacje o stronach WWW. Zazwyczaj żeby rozpoznać roboty, umieszcza się na każdej stronie jednopunktowy obrazek będący specjalnym linkiem. Człowiek nie jest w stanie dostrzec tego obrazka i nigdy nie wybierze tego linku - robot natomiast przechodzi najczęściej wszystkie linki na stronie - a więc też ten ukryty. Roboty działające w internecie powinny być zgodne ze standardem ([23]).

**Serwis WWW** - Zbiór zasobów dostępnych przez protokół HTTP połączonych w jedną logiczną całość.

**Sesja** - Zestaw akcji wykonanych przez klienta w jednym logicznym ciągu.

**SEE (Search Engine Enhancer)** - moduł ulepszający działanie wyszukiwarki

**Ścieżka** - Ciąg stron, które klient odwiedził w ramach jednej sesji za pomocą kolejnych kliknięć na linki dostępne na stronach (czyli bez używania przycisku wstecz tudzież zakładek w przeglądarce).

**Wyszukiwarka** - Narzędzie indeksujące zasoby serwisu WWW i wypisujące je według pewnego klucza związanego z zapytaniem zgłoszonym przez użytkownika.

# Bibliografia

## B.1. Site Improving Engine

[1] Cezary Kaliszyk *SIE — Site Improving Engine*, praca licencjacka na kierunku informatyka, 2003

[2] G. Andruszkiewicz, K. Ciebiera, M. Gozdalik, C. Kaliszyk, M. Srebrny. SIE — Intelligent Web Proxy Framework. In *4th International Conference on Web Engineering (ICWE'2004)*, volume 3140 of *Lecture Notes in Computer Science*, pages 373-385, 2004

[3] Radosław Borecki, Przemysław Marszałek, Michał Matusiak i Mateusz Srebrny. Moduł Search Engine Enhancer, Ucząca się wyszukiwarka zasobów serwisu WWW, 2003

[4] Karina Łuksza, Marta Łuksza, Janusz Dutkowski, Marcin Gozdalik. Dokumentacja modułu Adapter, 2003

[5] Bogusław Kluge, Łukasz Lew, Ewa Mąkosa, Dominik Sidorek. *Dokumentecja języka WTL*, praca licencjacka na kierunku informatyka, 2003

## B.2. Adaptive Web Mining

[6] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of SIGMOD–93*, pages 207–216, 1993.

[7] C. Anderson, P. Domingos, and D. Weld. Adaptive web navigation for wireless devices, 2001.

[8] C. Anderson, P. Domingos, and D. Weld. Relational markov models and their application to adaptive web navigation, 2002.

[9] C. R. Anderson. *A Machine Learning Approach to Web Personalization*. PhD thesis, University of Washington, 2002.

[10] R. Barrett and P. P. Maglio. Intermediaries: New places for producing and manipulating web content. In *World Wide Web*, 1999.

[11] R. Barrett, P. P. Maglio, and D. C. Kellem. How to personalize the web. In *Proceedings of the Conference on Human Factors in Computing Systems CHI'97*, 1997.

[12] I. V. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White. Visualization of navigation patterns on a web site using model-based clustering. In *Knowledge Discovery and Data Mining*, pages 280–284, 2000.

[13] S. Chakrabarti. *Mining the Web*. Morgan Kaufmann Publishers, San Francisco, 2003.

[14] R. Cooley. *Web Usage Mining: Discovery and Application of Interesting Patterns from Web Data*. PhD thesis, University of Minnesota, 2000.

[15] M. Deshpande and G. Karypis. Selective Markov Models for Predicting Web-Page Accesses, 2001.

[16] G. Karypis and E.-H. Han. Concept indexing: a fast dimensionality reduction algorithm with applications to document retrieval and categorization. Technical report tr-00-0016, University of Minnesota, 2000.

[17] B. Mobasher, H. Dai, and M. Tao. Discovery and evaluation of aggregate usage profiles for web personalization, 2002.

[18] M. Perkowitz. *Adaptive Web Sites: Cluster Mining and Conceptual Clustering for Index Page Synthesis*. PhD thesis, University of Washington, 2001.

[19] M. Perkowitz and O. Etzioni. Adaptive Web Sites: an AI Challenge. In *IJCAI (1)*, pages 16–23, 1997.

[20] M. Perkowitz and O. Etzioni. Towards adaptive Web sites: conceptual framework and case study. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1245–1258, 1999.

[21] P. Pirolli, J. Pitkow, and R. Rao. Silk from a sow's ear: Extracting usable structures from the web. In *CHI-96*, Vancouver, 1996.

[22] W3C. Web characterization activity. http://www.w3.org/WCA.

## B.3. Narzędzia i protokoły

[23] M. Koster, A Standard for Robot Exclusion, 1994
http://www.robotstxt.org/wc/norobots.html

[24] *HTML 4.01 Specification, http://www.w3.org/TR/html4*

[25] *RFC 2616 — HTTP 1.1 Specification*

[26] *RFC 2817 — Upgrading to TLS Within HTTP/1.1*

[27] *RFC 2818 — HTTP Over TLS*

[28] Xavier Leroy (with Damien Doligez, Jacques Garrigue, Didier Rémy and Jérôme Vouillon) *The Objective Caml system release 3.08, Documentation and user's manual* July 13, 2004, http://caml.inria.fr/ocaml/htmlman/index.html

[29] *Persistent client state HTTP cookies, http://wp.netscape.com/newsref/std/cookie_spec.html*

[30] Niels Provos. Libevent — An event notification library http://www.monkey.org/ provos/libevent/, November 2000

[31] J. Lemon. Kqueue: a generic and scalable event notification facility. In *Proceedings of the 2001 Annual Usenix Technical Conference*, June 2001

[32] Bruce Momjian, PostgreSQL — Introduction and Concepts, published by Addison Wesley, 2000

[33] Alain Frisch, Dokumentacja sterownika PostgreSQL dla OCamla (postgres.mli oraz programy przykładowe), 2001

[34] Abhishek Chandra and David Mosberger. Scalability of Linux event-dispatch mechanisms. In *Proceedings of the USENIX Annual Technical Conference*, pages 231-244, June 2001.

[35] Brice Goglin and Loïc Prylli. Performance Analysis of Remote File System Access over a High-Speed Local Network. In *Proceedings of the Workshop on Communication Architecture for Clusters (CAC'04), held in conjunction with the 18th IEEE IPDPS Conference*, Santa Fe, New Mexico, April 2004. IEEE Computer Society Press

[36] Michael Hicks, Stephanie Weirich, and Karl Crary. Safe and flexible dynamic linking of native code. In R. Harper, editor, *Types in Compilation: Third International Workshop, TIC 2000; Montreal, Canada, September 21, 2000; Revised Selected Papers*, volume 2071 of *Lecture Notes in Computer Science, pages 147-176. Springer*, 2001.

[37] David Mosberger and Tai Jin. httperf: A tool for measuring web server performance. In *First Workshop on Internet Server Performance*, 59-69. ACM, June 1998.

## B.4. Wyszukiwarki

[38] http://www.google.com/about.html

[39] Krzysztof Wecel, „Jak Google to robi?"
http://www.gazeta-it.pl/internet.html

[40] Phil Craven, „Google's PageRank Explained and how to make the most of it"
http://www.webworkshop.net/pagerank.html

[41] Strona domowa ht:Dig'a
http://www.htdig.org/

[42] http://searchenginewatch.com/webmasters/rank.html

[43] Wolfgang Sander-Beuermann, Mario Schomburg, „Internet Information Retrieval – The Further Development of Meta-Searchengine Technology"
http://www.isoc.org/inet98.html

[44] Annabel Pollock, Andrew Hockley, „What's Wrong with Internet Searching"
http://www.dlib.org/dlib.html

[45] Ben Shneiderman, „A User-Interface Framework for Text Searches"
http://www.cs.umd.edu/projects/hcil/

[46] Back to the basics of site searching
http://www.shorewalker.com/design/

[47] Shirley E. Kaiser, M.A., „Designing for Search Engines and Stars"
http://www.w3.org/

[48] Judith Lamont, „Finding the right piece of information, Structure and meaning improve search results"
http://www.kmworld.com/

[49] http://www.searchengineguide.org

[50] Raoul Masson, Gabriel Michel. Moteurs de recherche: vers une prise en compte de l'accessibilité pour les déficients visuels et les seniors. Proceedings of the *14th French-speaking conference on Human-computer interaction*, Pages: 235–238, 2002