Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Paweł Hryczuk
Nr albumu: 197864

Zarządzanie globalną historią w przeglądarce opartej na Mozilli

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dr Janiny Mincer-Daszkiewicz
Instytut Informatyki

Sierpień 2006

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora pracy

Świadomy 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 pracy

Streszczenie

Mozilla jest platformą aplikacji o szerokich zastosowaniach, szczególnie w zakresie oprogramowania sieciowego. Najbardziej znanym przykładem wykorzystania jej możliwości jest przeglądarka Firefox. Tematem niniejszej pracy jest jej rozszerzenie dostarczające nową implementację globalnej historii odwiedzanych stron WWW. W porównaniu do istniejącej jest ona znacznie bardziej efektywna i funkcjonalna. Historia przedstawiana jest w postaci drzewa, a nie listy. Celem było zwiększenie użyteczności tego zwykle niedocenianego elementu przeglądarki.

Praca składa się z dwóch części. W pierwszej opisane zostały składniki platformy oraz stosowane technologie. Dużo uwagi poświęcono budowie interfejsu użytkownika, komponentom i modelowi danych. Druga dotyczy rozszerzenia mechanizmu historii: projektu, architektury i implementacji. Na końcu przedstawiono porównanie z innymi rozwiązaniami.

Słowa kluczowe

Mozilla, Firefox, przeglądarka, platforma aplikacji, platforma programistyczna, WebMemo, rozszerzenie, wizualizacja WWW

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

D. Software
D.3 Programming languages
D.3.3 Language Constructs and Features (*Frameworks*)
D.2 Software Engineering
D.2.6 Programming Environments (*Programmer workbench*)

Tytuł pracy w języku angielskim

Global history management in a browser based on Mozilla

Spis treści

Rozdział 1

Wprowadzenie

1.1. Platforma aplikacji Mozilla

Wraz ze wzrostem mocy obliczeniowej komputerów zwiększa się złożoność oprogramowania. W konsekwencji zmieniają się też techniki tworzenia aplikacji. Budowanie od podstaw przy użyciu niskopoziomowych języków programowania staje się niepraktyczne. Nowoczesne rozwiązania coraz częściej opierają się na platformach aplikacji. Pozwala to na efektywne pisanie złożonego kodu z wykorzystaniem gotowych narzędzi i bibliotek.

Mozilla nie jest tak znaną platformą aplikacji jak .NET albo Java. Wyróżnia się określonym obszarem zastosowań, którym są przede wszystkim aplikacje sieciowe. Spośród nich największy sukces odniosła przeglądarka Firefox, przewyższająca konkurencję pod wieloma względami.

Jedną z charakterystycznych cech Mozilli jest wbudowany mechanizm rozszerzeń. W połączeniu z elastyczną architekturą platformy umożliwia w prosty sposób dodawanie funkcjonalności do jej aplikacji. Firefox nie byłby drugą pod względem popularności przeglądarką na świecie bez rozszerzeń takich jak *Adblock* [34] czy *FlashGot* [38].

1.2. Historia w przeglądarce internetowej

Zarządzanie globalną historią w przeglądarce polega na zapisywaniu adresów odwiedzanych stron internetowych, a następnie prezentowaniu ich użytkownikowi. Mimo że istnieje wiele zaawansowanych metod wizualizacji struktury WWW [2], we wszystkich współczesnych przeglądarkach do tego celu wykorzystuje się zwykłą listę. Nic dziwnego, że w praktyce mechanizm ten jest bardzo rzadko używany.

Dodatkowym problemem w Firefoksie jest ograniczenie długości tej listy do kilku ostatnich dni, co jest konsekwencją bardzo nieefektywnej bazy danych. Potrzeba nowej implementacji mechanizmu obsługi historii jest oczywista.

1.3. Zaawansowana obsługa historii w Firefoksie

Tematem niniejszej pracy jest oparte na platformie Mozilla rozszerzenie Firefoksa. Dwa cele projektu o nazwie WebMemo to efektywna obsługa historii przeglądarki oraz jej wizualizacja. Pierwszy zostanie zrealizowany przy pomocy nowego systemu bazy danych, a wypełnieniem drugiego będzie implementacja drzewa historii.

1.4. Plan pracy

Praca składa się z dwóch części. Pierwsza (rozdziały 2-4) jest opisem platformy. Rozdział 2 to krótka charakterystyka Mozilli, opis historii projektu i jego składników. W rozdziale 3 omówiono architekturę oraz technologie platformy. Dokładny opis budowy interfejsu, komponentów i modelu danych znajduje się w rozdziale 4.

Druga część (rozdziały 5 i 6) dotyczy rozszerzenia obsługi historii. Na początku przedstawiono wady istniejącego rozwiązania, następnie cel i wymagania projektowe WebMemo, architekturę i implementację, wreszcie jego możliwości i ograniczenia. Ostatni rozdział zawiera opis praktycznych doświadczeń, aktualnego stanu projektu oraz perspektyw dalszego rozwoju. W dodatkach znajdują się kolejno: instrukcja kompilacji, zrzuty ekranu obrazujące interfejs użytkownika oraz spis plików znajdujących się na załączonej płycie CD.

Rozdział 2

Od przeglądarki do platformy aplikacji

Słowo *Mozilla* ma kilka znaczeń. Przede wszystkim jest nazwą projektu o publicznym dostępie do kodu źródłowego (open-source), którego celem jest rozwój wieloplatformowego oprogramowania klienckiego dla Internetu [24]. Niekiedy odnosi się do związanej z nim organizacji i społeczności internetowej `mozilla.org`. Jednak najczęściej jest błędnie utożsamiane z nazwą przeglądarki internetowej.

Podstawą projektu Mozilla jest platforma aplikacji (ang. *Mozilla Application Framework*). W oparciu o nią powstała przeglądarka Firefox, podobnie jak klient pocztowy Thunderbird czy komunikator ChatZilla. Przykładem aplikacji niezwiązanej z Internetem jest kalendarz Sunbird.

Platforma Mozilla jest wykorzystywana również w innych projektach. Wywodzi się z niej linuksowa przeglądarka *Epiphany* [37], która jest częścią GNOME Desktop Suite. Istnieje też wiele zastosowań komercyjnych, jak na przykład środowisko programistyczne dla języków skryptowych *ActiveState Komodo* [33].

W niniejszej pracy Mozilla oznacza domyślnie platformę aplikacji.

2.1. Historia i rozwój projektu

Nazwa projektu pochodzi od „Mosaic Killer", ponieważ tym miała być przeglądarka firmy Netscape dla popularnej wówczas NCSA Mosaic. Oficjalnie Mozilla była nazwą kodową programu Netscape Navigator, z czasem stała się też charakterystyczną maskotką przypominającą dinozaura oraz jednym z symboli firmy Netscape. Na początku 1998 roku w Netscape Communications Corporation podjęto odważną i kontrowersyjną decyzję o zrzeczeniu się praw autorskich do kodu przeglądarki[1]. Mozilla narodziła się w dniu publikacji kodu źródłowego, 1 kwietnia 1998 roku [16].

Chociaż pracownicy firmy Netscape nadal pracowali nad projektem, Mozilla uniezależniała się wraz z rozwojem społeczności *Mozilla Organization*. Wkrótce zyskała nowych sponsorów (m.in. firmę Red Hat), którym zależało na rozwoju wolnodostępnej przeglądarki internetowej. W 2003 roku America Online (obecny właściciel firmy Netscape) wycofała się z Mozilli. W miejsce *Mozilla Organization* powołano bardziej formalną *Mozilla Foundation*.

[1] Decyzja ta była w znacznej mierze spowodowana utratą przewagi rynkowej nad konkurencyjnym Microsoft Internet Explorer, który swój sukces zawdzięczał zakupionej od Spyglass (następcy NCSA) licencji na kod źródłowy przeglądarki Mosaic [30].

2.2. Wersje Mozilli

Mozilla 1.0 została oficjalnie wydana 5 czerwca 2002 roku. Pewna część interfejsów platformy została oznaczona jako zamrożone (*frozen*), ogłoszono standard XUL 1.0 oraz standard XBL. Celem tych decyzji było zapewnienie zgodności z przyszłymi wersjami Mozilli 1.x (zmiany będą możliwe dopiero w wersji 2.0). Ten fakt ma duże znaczenie dla twórców rozszerzeń, którzy nie biorą udziału w pracach nad samą Mozillą.

Początkowo pakiet *Mozilla Application Suite* był rozwijany jako całość, jednak wkrótce zauważono, że forma ta nie jest odpowiednia dla projektu open-source. Dlatego skoncentrowano się na dwóch najważniejszych produktach: przeglądarce Firefox i kliencie pocztowym Thunderbird. Aktualną wersją Firefoksa jest 1.5, a rozwijaną 2.0[2]. Podstawą dla wszystkich aplikacji platformy jest Mozilla 1.8.

2.3. Przegląd składników

2.3.1. Produkty Mozilli

Programiści, którzy stworzyli platformę aplikacji, obecnie rozwijają oparte na niej aplikacje. Dzięki bardzo dobrej znajomości budowy i działania Mozilli potrafią najlepiej wykorzystywać jej możliwości. Co więcej, w zależności od potrzeb mogą modyfikować samą platformę. W ten sposób ewoluuje Mozilla.

Firefox[3]
: Najważniejszy i obecnie najbardziej intensywnie rozwijany projekt jest nową implementacją wcześniejszego *Mozilla Navigator*. Celem jest stworzenie efektywnej, wolnodostępnej i wieloplatformowej przeglądarki stron WWW. Firefox charakteryzuje się wysokim poziomem bezpieczeństwa, dobrą zgodnością ze standardami W3C, łatwością w użyciu oraz innowacyjnością. Obsługuje m.in. przeglądanie w panelach, automatyczną aktualizację, kanały RSS i szybki dostęp do wyszukiwarek internetowych. Spośród wszystkich projektów Mozilli posiada zdecydowanie najwięcej rozszerzeń [31].

Thunderbird
: Rozbudowany klient pocztowy obsługujący wszystko, czego po takiej aplikacji można się spodziewać (m.in. protokoły IMAP i POP3, listy w formacie HTML, sprawdzanie pisowni). Ważną zaletą jest również dynamicznie uczący się filtr antyspamowy. Dla użytkowników innych programów pocztowych istotne są możliwość importu i eksportu książki adresowej oraz przechowywanie listów w standardowym formacie *mbox*. Podobnie jak w przypadku Firefoksa dużo uwagi poświęcono bezpieczeństwu, jest też dostępnych wiele rozszerzeń.

SeaMonkey
: Pakiet zawierający przeglądarkę, klienta pocztowego, klienta grup dyskusyjnych oraz edytor HTML. SeaMonkey jest kontynuacją *Mozilla Application Suite*, jednak nie jest tak priorytetowym projektem jak jego poprzednik. Ponieważ zawiera wszystkie cechy aktualnej wersji platformy Mozilla, jest nadal interesującą propozycją dla wielu użytkowników.

[2] Nowa wersja przeglądarki nie będzie znacząco różniła się od obecnej, jednak przy numeracji przeważyły względy marketingowe [18].

[3] Nazwy projektów Mozilli zostały zaczerpnięte ze świata zwierząt, w tym przypadku jest to popularne określenie pandy małej *Ailurus fulgens*.

Bugzilla
: Przeznaczony dla administratorów i zespołów programistów system śledzenia błędów. Jest znacznie bardziej zaawansowany od konkurencyjnych bezpłatnych aplikacji i dlatego zdobył dużą popularność. Stosunkowo niezależny od platformy, powstał jeszcze w Netscape jako narzędzie programistyczne do rozwijania Mozilli. Tę rolę pełni nadal, podobnie jak związane z nim Tinderbox i Bonsai (opis tych narzędzi znajduje się w [26]).

Sunbird[4]
: Kalendarz i aplikacja do zarządzania informacją osobistą (ang. *Personal Information Management*) charakteryzująca się wysokim stopniem integracji z platformą. Sunbird powstał jako przykład zastosowania technologii Mozilli do stworzenia aplikacji niezwiązanej z typowymi usługami internetowymi. Ma wskazywać na wszechstronność potencjalnych zastosowań platformy i zachęcać do wykorzystywania jej we własnych projektach. Dla programistów stanowi rozbudowany przykład kodu korzystającego z Mozilli. Aktualna wersja 0.3 implementuje jedynie podstawową funkcjonalność terminarza.

2.3.2. Mozilla jako platforma aplikacji

Pozostali programiści widzą w Mozilli cenne narzędzie, które mogą wykorzystywać we własnej pracy. Dla nich ważne są nie tyle konkretne produkty, co sama platforma. Możliwe są trzy rodzaje zastosowań.

- Najczęstszym przypadkiem jest zagnieżdżanie (ang. *embedding*) silnika graficznego Gecko. Jest to najważniejszy element platformy Mozilla, który służy do interpretacji (ang. *rendering*) stron WWW. Aplikacja komunikuje się z Gecko za pomocą zdefiniowanego w tym celu API. Przykładem może być niewielka przeglądarka *K-Meleon* [43], która posiada własny interfejs, a do prezentowania stron używa Gecko.

- Druga możliwość to rozwijanie aplikacji w oparciu o platformę Mozilla, co wymaga ścisłej z nią integracji. Oprócz silnika graficznego dostępnych jest wiele narzędzi i gotowych składników typowych aplikacji, nie tylko przeglądarki czy klienta pocztowego. Na przykład w *Crocodile Clips* powstały w ten sposób komercyjne programy do modelowania.

- Trzecim zastosowaniem jest tworzenie rozszerzeń. Stosunkowo proste dołączanie nowego kodu było zawsze ważnym wymaganiem projektowym platformy Mozilla. Istnieje możliwość dokonywania znacznych modyfikacji (szczególnie interfejsu użytkownika) za pomocą samych języków skryptowych. Firefox posiada obecnie bardzo wiele darmowych rozszerzeń.

Ponieważ WebMemo jest rozszerzeniem Firefoksa, dalsza część pracy koncentruje się na ostatnim sposobie integracji z Mozillą.

2.4. Mozilla a platforma Microsoftu

Koncepcja platformy aplikacji nie jest nowa, także technologie stosowane w Mozilli są powszechnie znane. A jednak zaskakuje wzajemne podobieństwo niezależnych platform Mozilli i Microsoftu, szczególnie biorąc pod uwagę zapowiadany Windows Vista. Mogłoby się wręcz wydawać, że różnice sprowadzają się do nazw technologii.

[4] Znany też pod nazwą *Lightning* jako rozszerzenie Thunderbirda.

- COM (ang. *Component Object Model*) Microsoftu spełnia tę samą funkcję co XPCOM Mozilli (porównanie znajduje się w podrozdziale 4.2).
- Języki komponentów .NET (C#, JScript.NET) są bardzo podobne do tych w Mozilli (C++, JavaScript).
- Funkcję Gecko w Windows Vista będzie spełniał *Avalon*, a biblioteki sieciowej Necko – biblioteka *Indigo*.
- Interfejs użytkownika w Windows Vista będzie opisywany w języku XAML, różniącym się od XUL Mozilli wyłącznie składnią (oba są oparte na XML).
- Dane będą opisywane w XML Schema, podczas gdy Mozilla używa do tego celu RDF (również pochodzącego od XML).

Opis technologii używanych w Mozilli znajduje się w następnym rozdziale. W tym miejscu należy zauważyć, że obie platformy aplikacji łączy bardzo wiele[5]. Jest to dodatkowa motywacja do zajmowania się tak jedną, jak i drugą.

2.4.1. Internet Explorer

Internet Explorer także posiada możliwość dodawania rozszerzeń, tzw. *add-ons*. Wykorzystywane są punkty rozszerzeń (ang. *extension points*), jednak taki sposób dołączania kodu nie dorównuje elastyczności, z jaką mamy do czynienia w przypadku Mozilli. Microsoft, nie chcąc udostępnić kodu źródłowego przeglądarki, musiał ograniczyć się do takiego rozwiązania. Zatem stosunkowo łatwo można dodać nowy pasek narzędzi, ale na przykład zaimplementowanie przeglądania w panelach jest skomplikowane, chociaż nadal możliwe. Polityka Microsoftu polega raczej na dodawaniu bardziej pożądanych cech w kolejnych wersjach Internet Explorera, niż na wspieraniu rozwoju niezależnych rozszerzeń[6]. Programistów zniechęca też bardzo duża liczba błędów zabezpieczeń przed wirusami i programami typu *spyware*, co może okazać się szczególnie niebezpieczne w połączeniu z używaniem rozszerzeń. Poza tym brakuje prostego mechanizmu zarządzania rozszerzeniami, jaki można znaleźć w Mozilli.

Podobnie jak Mozilla pozwala na zagnieżdżanie Gecko, tak Internet Explorer umożliwia niezależne wykorzystywanie komponentu *WebBrowser*. W ten sposób użytkownicy Firefoksa w systemie Windows mogą otwierać panele używające silnika graficznego Microsoftu do prezentowania stron WWW (rozszerzenie *IE View* [42]). Oczywiście w tym przypadku integracja komponentu z przeglądarką jest niewielka.

[5] Szczegółowy opis oraz porównanie przykładowego kodu źródłowego w technologiach Microsoftu i Mozilli znajdują się w [8].

[6] Wspomniane przeglądanie w panelach jest tego dobrym przykładem. Użytkownicy wersji 6.0 mają do wyboru kilka rozszerzeń oferujących tę ważną funkcjonalność (m.in. *Maxthon* i *Avant Browser*), która została zaimplementowana dopiero w wersji 7.0 przeglądarki.

Rozdział 3

Architektura Mozilli

Poprzedni rozdział zawiera stwierdzenie, że Mozilla jest platformą aplikacji. Tematem niniejszego jest rozwinięcie tej tezy oraz przedstawienie i opis architektury systemu.

Najogólniej można powiedzieć, że Mozilla jest połączeniem mechanizmu przetwarzania dokumentów XML, interpretera języków skryptowych oraz komponentów. Platforma przeznaczona jest przede wszystkim do tworzenia interaktywnych aplikacji, w których komunikacja z użytkownikiem odgrywa szczególną rolę. Przykładem takiego zastosowania jest przeglądarka internetowa.

Pod względem długości kodu źródłowego Mozilla jest bardzo dużym projektem, jednym z kilku największych w świecie open-source. Dwa razy większa od jądra Linuksa jest wielkości projektu GNOME 2.0 łącznie ze standardowymi aplikacjami. Posiada ponad 1000 komponentów i ponad 1000 interfejsów [9]. Kod Mozilli został napisany bardzo starannie, z użyciem autorskiego zestawu makr, aby platformie zapewnić przenośność i ułatwić współpracę programistom z całego świata.

3.1. Diagram architektury

Diagram 3.1 przedstawia strukturę platformy Mozilla. Każdy prostokąt odpowiada jednej technologii, najczęściej specyficznej dla Mozilli, opartej na jednym lub kilku standardach[1]. Mniejsze, połączone kaskadowo prostokąty oznaczają pliki używane przez Mozillę. Poszczególne elementy są od siebie zależne, diagram prezentuje jedynie koncepcję platformy.

Widoczny jest podział na dwie części. Po jednej stronie znajdują się elementy związane z interfejsem użytkownika, jak DOM, CSS i URL. Są to dobrze znane standardy WWW, które tu w innym kontekście pełnią podobną rolę. Znaczna ich część opiera się na języku XML, z tego powodu ważnym dla Mozilli. Po drugiej stronie położone są technologie niskopoziomowe zapewniające m.in. przenośność i bezpieczeństwo, a także obsługę komponentów. Połączenie obu części diagramu następuje przez konwersję URL komponentu na jego *Contract ID* w kodzie JavaScript albo bezpośrednio przez użycie RDF.

3.2. Przegląd technologii

W celu lepszego zrozumienia architektury Mozilli w tym podrozdziale zostały przedstawione technologie wchodzące w skład platformy. Dokładny opis ich znaczenia i wzajemnych relacji znajduje się w dalszej części pracy.

[1] Niektóre prostokąty (np. JVM i JNI) odnoszą się do powszechnie znanych standardów. Technologie związane z Javą nie są istotne z punktu widzenia niniejszej pracy.

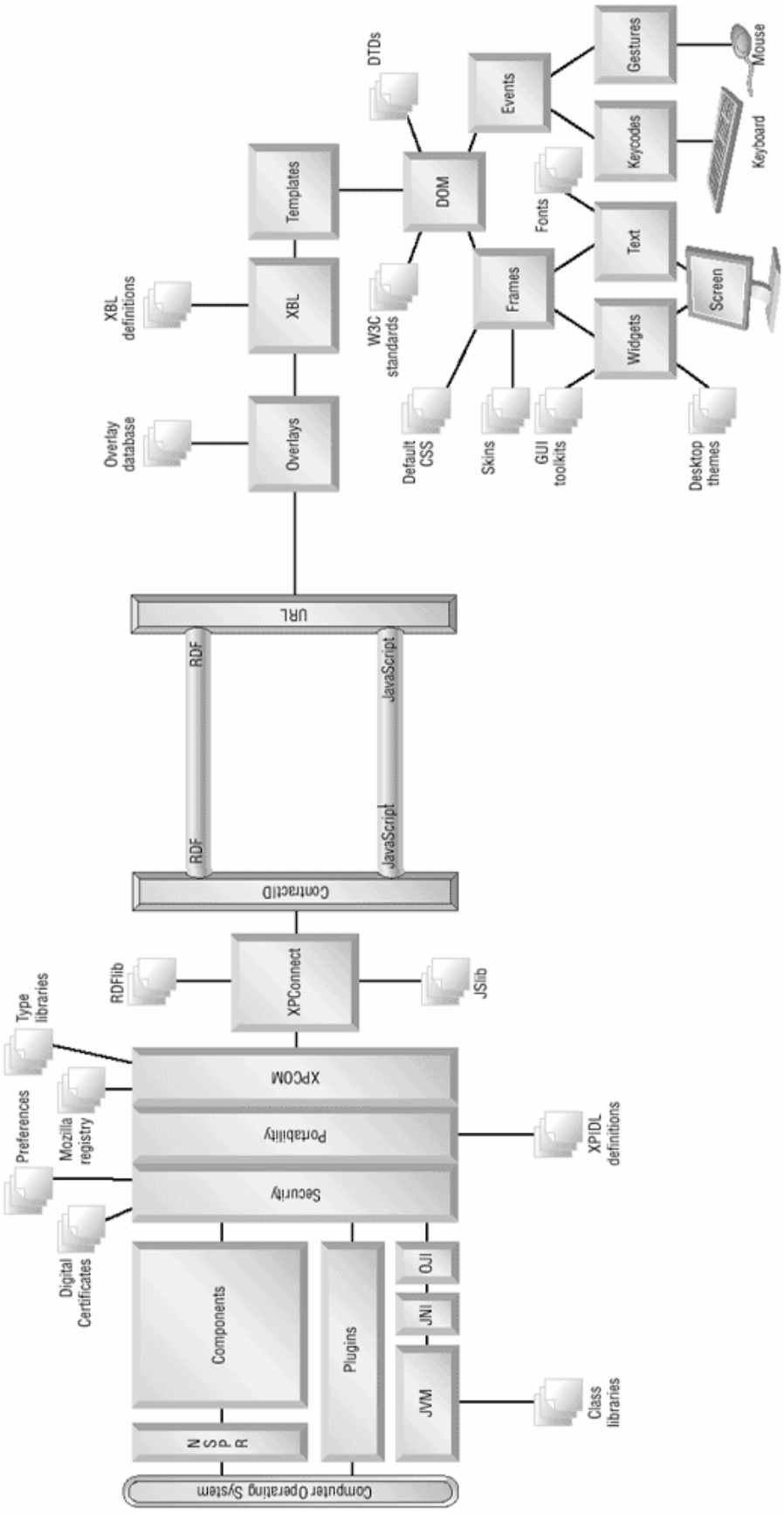


Diagram 3.1: Architektura platformy Mozilla [9]

3.2.1. Interfejs użytkownika

XUL

XUL (ang. *XML User Interface Language*) jest językiem opisu interfejsu użytkownika w postaci dokumentu (w przeciwieństwie do programu). Umożliwia szybkie tworzenie przenośnych interfejsów oraz obsługuje złożone kontrolki, takie jak drzewa. Jest ściśle powiązany z kolejnymi technologiami.

XBL

Dzięki XBL (ang. *XML Binding Language* albo *Extensible Bindings Language*) możliwe jest modyfikowanie istniejących kontrolek XUL oraz dodawanie nowych. Ta elastyczność wyróżnia programowanie interfejsów w Mozilli od metod stosowanych w innych środowiskach i językach programowania. Prawie wszystkie standardowe kontrolki zostały zaimplementowane w XBL.

CSS

CSS (ang. *Cascading Style Sheets*) jest dobrze znanym językiem służącym do opisu prezentacji dokumentów XML. W Mozilli jest wykorzystywany do stylizacji interfejsu użytkownika, dzięki czemu tworzenie motywów (ang. *themes*) jest proste i nie wymaga wiedzy informatycznej.

DTD

Znajomość programowania nie jest konieczna także do lokalizacji aplikacji. DTD (ang. *Document Type Definition*) jest używany do rozwijania makr umieszczanych w skryptach XUL, których głównym zastosowaniem jest tłumaczenie etykiet i napisów.

JavaScript[2]

Połączenie interfejsu z logiką aplikacji przebiega za pośrednictwem języka JavaScript, którego kod jest wywoływany podczas obsługi zdarzeń (np. zamknięcie okna). Komponenty XPCOM mogą być również implementowane przy użyciu JavaScriptu. Zastosowanie języka skryptowego jest często znacznie wygodniejsze niż programowanie w C++.

3.2.2. Technologie niskopoziomowe

RDF

RDF (ang. *Resource Description Framework*) jest językiem opartym na XML służącym do opisu relacji. W Mozilli wykorzystywany jest do zarządzania danymi, między innymi zakładek i historii. W tym przypadku stanowi warstwę pośrednią pomiędzy bazą danych a interfejsem użytkownika.

XPCOM

Technologia XPCOM (ang. *Cross-Platform Component Object Model*) polega na podziale kodu aplikacji na mniejsze komponenty. Konsekwencje wynikające z takiego modelu programowania zostały przedstawione dalej. W skład XPCOM wchodzi kilka ważnych bibliotek (np. do zarządzania pamięcią) i narzędzi związanych z komponentami.

XPIDL

Każdy komponent posiada interfejs napisany w języku XPIDL (ang. *Cross-Platform Interface Description Language*), za pośrednictwem którego uzyskiwany jest dostęp do

[2] Warto zaznaczyć, że JavaScript powstał w firmie Netscape w 1995 roku.

metod i atrybutów komponentu. XPIDL powstał jako dostosowanie języka IDL do potrzeb platformy Mozilla.

XPConnect
XPConnect (ang. *Cross-Platform Connect*) umożliwia odwoływanie się do komponentów XPCOM z poziomu innych języków programowania. Początkowo był to jedynie JavaScript, obecnie także Java i Python, a wkrótce Perl i Ruby. Możliwe jest także programowanie samych komponentów w JavaScript.

XPInstall
XPInstall (ang. *Cross-Platform Install*) jest technologią instalowania rozszerzeń dla aplikacji Mozilli, takich jak Firefox i Thunderbird. Definiuje strukturę archiwum oraz format pliku RDF opisującego rozszerzenie (tzw. *manifest*).

XULRunner
XULRunner zajmuje się instalacją, aktualizacją i deinstalacją aplikacji z systemu operacyjnego. Do jego zadań należy też początkowe ładowanie (ang. *bootstrapping*) aplikacji.

3.2.3. Technologie wspierane przez Mozillę

Ponadto Mozilla posiada wsparcie dla RSS (ang. *Really Simple Syndication*), XSLT (ang. *Extensible Stylesheet Language Transformations*), XForms (ang. *XML Forms*), SVG (ang. *Scalable Vector Graphics*), MathML oraz Web Services (SOAP, XML-RPC, WSDL). Te składniki platformy nie występują w dalszej części pracy, zostały podane dla pełności opisu.

3.2.4. Podsumowanie

Technologie będące standardami W3C, jak CSS i JavaScript, pełnią w Mozilli podwójną rolę. Po pierwsze tworzą interfejs użytkownika aplikacji platformy, po drugie są stosowane na stronach WWW interpretowanych przez Gecko. Na stronach internetowych mogą być wykorzystywane również technologie Mozilli (np. język XUL).

3.3. Platforma aplikacji

> Platforma aplikacji jest to termin używany w odniesieniu do zestawu bibliotek lub klas, używanych w danym systemie do implementacji aplikacji o standardowej strukturze. Poprzez skompletowanie dużej ilości kodu wielokrotnego użytku zmniejsza się czas pracy programisty, który nie musi powtarzać standardowego kodu w każdej nowej aplikacji [29].

Tak rozumianą strukturę aplikacji w przypadku Mozilli tworzą: XUL, JavaScript, RDF oraz XPCOM. Te cztery technologie zostały obszernie opisane w kolejnym rozdziale. Pozostała część niniejszego dotyczy budowy platformy Mozilla.

3.3.1. Od DHTML do XPFE

Ciekawy jest fakt, że zgodnie z podaną definicją każda przeglądarka internetowa jest platformą aplikacji. DHTML (ang. *Dynamic HTML*) daje możliwość programowania stron WWW

jako aplikacji, dla których środowiskiem jest okno przeglądarki. Taki sposób programowania interfejsu użytkownika jest bardzo wygodny i zyskał dużą popularność[3].

DHTML składa się ze statycznego HTML, języka skryptowego JavaScript oraz używanego do prezentacji danych CSS. Jak wiadomo HTML opiera się na koncepcji hypertekstu i operuje pojęciami znanymi z edytorów tekstowych (takimi jak *nagłówek*, *paragraf*, *czcionka*). Używanie go do opisu interfejsu użytkownika jest nieintuicyjne i niezgodne z jego przeznaczeniem, a przez to nieefektywne.

Mozilla XPFE (ang. *Cross-Platform Front End*) łączy duże możliwości JavaScript z CSS i specjalnie zaprojektowanym językiem XUL. Podobny do HTML, różni się głównie zestawem pojęć (np. *okno*, *przycisk*, *menu*). W ten sposób opisany jest interfejs samej Mozilli, a także jej aplikacji. To dlatego tak dobrze nadaje się ona do tworzenia wizualnych i interaktywnych aplikacji.

3.3.2. Rola języka XML

Język XML jest jednym z kluczowych elementów platformy Mozilla, ponieważ na nim opiera się wiele technologii: XUL, HTML, XBL, RDF, a także SVG, RSS i MathML. Tak więc służy on do opisywania interfejsu, interpretowania stron WWW, przechowywania danych, a także przedstawiania grafiki wektorowej i wzorów matematycznych. W warstwie XPCOM istnieje grupa komponentów wykorzystujących standard DOM (ang. *Document Object Model*) do wykonywania operacji na dokumentach XML. Języki jemu pokrewne mają wiele wspólnych zalet, przede wszystkim elastyczność i otwartość, hierarchiczną strukturę oraz przenośność (dokumenty mogą być kodowane w wielu formatach, m.in. w stosowanym w platformie UTF).

3.3.3. Model warstwowy

Poszczególne elementy platformy tworzą model warstwowy przedstawiony na diagramie 3.2. Zaznaczono na nim położenie i powiązania głównych technologii specyficznych dla platformy.

Najniższa warstwa jest odpowiedzialna za implementację i zarządzanie komponentami XPCOM. Tutaj znajdują się m.in. obiekty zapewniające przenośność platformy poprzez abstrakcję od konkretnego systemu operacyjnego. Cała warstwa została napisana w języku C++, chociaż na tym poziomie możliwe jest też wykorzystywanie JavaScriptu. Każdy komponent posiada interfejs XPIDL, przez który za pośrednictwem warstwy XPConnect łączy się z innymi komponentami oraz warstwami systemu. Na wyższych warstwach używany jest język skryptowy, najczęściej JavaScript. Zaletami są wtedy: łatwość pisania kodu (np. brak silnych typów), brak konieczności kompilacji, przejrzystość oraz wieloplatformowość.

Ostatnia warstwa związana jest z interfejsem użytkownika i obsługą języków XML. Ponieważ bardziej skomplikowane operacje (takie jak konwersja plików XML do postaci zgodnej ze standardem DOM) są realizowane przez komponenty warstwy XPCOM, na tym poziomie wystarczy sam JavaScript. Dwa najważniejsze zadania to przetwarzanie dokumentów stron internetowych (HTML) oraz interfejsu użytkownika (XUL). Aby zwiększyć bezpieczeństwo dodatkowo wprowadzane są ograniczenia uprawnień.

Istnieją też inne sposoby łączenia komponentów z dokumentami XUL. Najważniejszym jest używanie szablonów (ang. *XUL Templates*). Na poziomie warstwy XPCOM dane są formatowane zgodnie ze standardem RDF, a następnie przetwarzane automatycznie – zgodnie z regułami odpowiedniego szablonu. Innym sposobem jest rozszerzanie funkcjonalności języka XUL przy pomocy XBL.

[3] Obecnie jest często stosowany nawet lokalnie, tzn. klient i serwer HTTP są uruchomiane na tym samym komputerze.

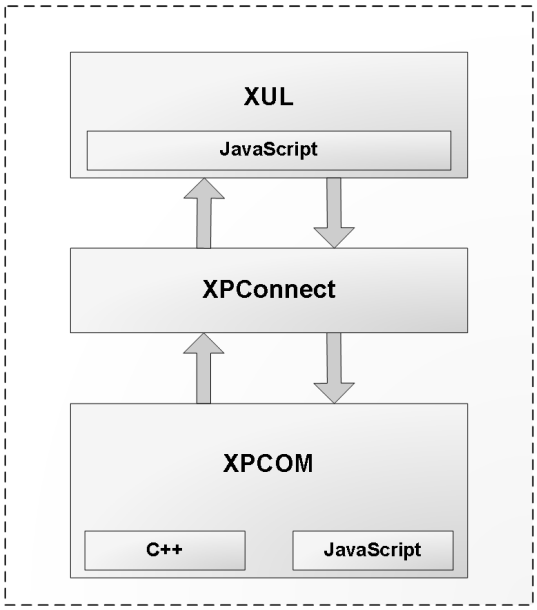


Diagram 3.2: Model warstwowy platformy Mozilla (por. [4])

3.3.4. Powłoka aplikacji

Jak dotąd Mozilla bardziej sprawia wrażenie zbioru bibliotek niż platformy. Gdyby tak było, programista musiałby ręcznie uruchamiać swoje aplikacje, zarządzać cyklem ich życia, dostępem do bibliotek, a także procesem instalacji. Dlatego ważnym elementem platformy jest powłoka aplikacji (ang. *application shell*), dzięki której możliwe jest otrzymanie programu wykonywalnego nawet z kilku dokumentów XML. Jednym z elementów tej powłoki jest XPInstall.

Rozdział 4

Opis platformy

4.1. Projektowanie interfejsu użytkownika

4.1.1. Budowa i wygląd interfejsu

Sposób budowy interfejsu użytkownika wyróżnia platformę Mozilla od innych środowisk i języków programowania. Tradycyjna metoda tworzenia interfejsu polega na wykorzystywaniu odpowiednich narzędzi GUI, którymi na przykład w Javie są AWT, SWT i Swing. Interfejs i logika aplikacji programowane są w tym samym języku, razem kompilowane i testowane. W konsekwencji programista Javy jest ograniczony w wyborze biblioteki GUI do jednej z trzech wymienionych, których z kolei nie mógłby wykorzystać w innym języku.

Java i Mozilla posiadają wiele cech wspólnych, a jedną z nich jest wieloplatformowość. Systemy operacyjne różnią się budową i możliwościami interfejsów graficznych, co stwarza dodatkowy problem dla przenośnych aplikacji. W celu zachowania jednolitego wyglądu interfejsu Java posługuje się niezależnymi od systemu tematami. Dlatego jej aplikacje wyróżniają się w stosunku do macierzystych programów danego systemu. Alternatywną metodą jest dopasowywanie wyglądu interfejsu do bieżącego środowiska i wykorzystywanie jego specyficznych własności, co jest jednak bardziej pracochłonne. Mozilla wspiera oba rozwiązania, a wybór pomiędzy nimi pozostawia programiście [4].

Jest to przykład bardzo dużej elastyczności interfejsu Mozilli, która jest rezultatem innowacyjnego podejścia do problemu. Interfejs użytkownika jest p r o j e k t o w a n y w języku XUL służącym wyłącznie do tego celu. W odróżnieniu od programu opis interfejsu jest dokumentem interpretowanym w czasie działania aplikacji. Ma to wiele zalet:

- elastyczność – aplikacja może być napisana w dowolnym języku współpracującym z platformą;
- oddzielenie od logiki – zmiany w interfejsie nie powodują konieczności zmian w logice;
- brak konieczności kompilacji – wygoda dla programisty i możliwość modyfikowania interfejsu przez użytkownika;
- prosta struktura i przejrzystość – zalety wynikające z użycia XML;
- możliwość lokalizacji – dzięki umieszczaniu napisów w encjach (rozwijanych w plikach DTD).

Interfejs użytkownika składa się z trzech typów dokumentów. W języku XUL określa się kontrolki, ich atrybuty, strukturę i układ (ang. *layout*). Standardowy wygląd kontrolek może być modyfikowany w dokumentach CSS. Język CSS jest stosowany zgodnie ze specyfikacją W3C, przy czym platforma definiuje prawie 200 dodatkowych własności[1]. Jego rola polega więc na stylizowaniu aplikacji.

Dokumenty XUL, CSS i DTD wchodzą w skład aplikacji, co oznacza, że użytkownicy mogą je dowolnie modyfikować. Budowa pakietów umożliwia dodawanie nowych dokumentów odpowiadających istniejącym [9]. W ten sposób instaluje się zmodyfikowane pliki CSS jako tematy (ang. *themes*), a przez dokumenty DTD lokalizuje się aplikacje. Nie wymaga to znajomości kodu źródłowego i jest wyrazem elastyczności w budowie interfejsu użytkownika.

4.1.2. Obsługa interfejsu

W tradycyjnej metodzie obsługi interfejsu instaluje się funkcję (klasę) oczekującą na określone zdarzenie, która następnie wywołuje odpowiednie metody logiki systemu. Deklaratywny język XUL nie może być użyty w ten sposób, dlatego wykorzystuje się JavaScript. Zagnieżdżanie skryptów w dokumentach XUL jest identyczne jak w stronach HTML, a interakcja z użytkownikiem przebiega w podobny sposób.

Istnieje też mechanizm instalowania funkcji obsługi zdarzeń (ang. *event handlers*), co zwykle ma miejsce podczas inicjowania danego elementu interfejsu (np. okna dialogowego). W kodzie JavaScript można odczytywać oraz modyfikować atrybuty kontrolek, które zawierają parametry zdarzeń.

Skrypty mogłyby ograniczać się do wywoływania metod komponentów XPCOM, jednak często wygodniej jest zaimplementować część funkcjonalności w języku JavaScript. Najlepszym tego przykładem jest plik Firefoksa `browser.js` mający ponad 6000 wierszy kodu. Oczywiście tam gdzie kluczowa jest wydajność wykorzystuje się C++.

4.1.3. Rozszerzanie interfejsu

Tradycyjny sposób programowania interfejsu użytkownika nie umożliwia modyfikacji kontrolek, ponieważ są one zdefiniowane w kodzie źródłowym biblioteki GUI. Natomiast w Mozilli język XBL opisuje dostępne kontrolki XUL oraz pozwala na modyfikację ich metod i atrybutów.

Relacja pomiędzy XBL a XUL jest podobna do relacji między C++ i C. XBL jest wysokopoziomowym językiem obiektowym, w którym możliwe jest m.in. dziedziczenie, agregacja i używanie interfejsów[2]. Mimo to została zachowana strukturalna zgodność z XML, a definicje metod zapisywane są jako wstawki JavaScript. Odpowiednikiem klasy jest powiązanie (ang. *binding*), a obiektu – zastosowanie powiązania do danego znacznika XUL (co jest definiowane w opisującym styl dokumencie CSS).

XBL umożliwia dodawanie powiązań, a nie kontrolek interfejsu. Programowanie nowych kontrolek jest w rzeczywistości modyfikowaniem sposobu działania i wyglądu istniejących. Ze względu na ścisłą integrację z pozostałymi technologiami Mozilli otrzymanie zupełnie dowolnej kontrolki nie jest możliwe. Zaletą XBL jest prostota opisu, a nie siła wyrazu.

[1] Jest to konieczne, aby móc wyrażać cechy interfejsu użytkownika w języku oryginalnie zaprojektowanym do tworzenia stron WWW.

[2] Szczegółowy opis cech obiektowych XBL znajduje się w [9].

4.2. Komponenty XPCOM

XPCOM jest jednym z głównych składników, dzięki którym Mozilla może nazywać się platformą aplikacji. Pod wieloma względami jest bardzo podobny do Microsoft COM (MSCOM). Różnice wynikają z odmiennych założeń: MSCOM powstał jako platforma ogólnego zastosowania, natomiast rozwój XPCOM odbywał się równolegle z rozwojem przeglądarki – a potem platformy – Mozilla.

4.2.1. Programowanie komponentowe

Programowanie jest tym trudniejsze, im dłuższy jest kod źródłowy. Podstawową zasadą jest podział większego problemu na mniejsze podproblemy – czy to w programowaniu strukturalnym, czy obiektowym. Wraz z dalszym rozwojem aplikacji technika ta staje się niewystarczająca, ponieważ zwiększa się wzajemne powiązanie klas. W programowaniu komponentowym implementacja zostaje oddzielona od publicznego interfejsu[3].

Interfejsy komponentów

Niektóre nowoczesne języki programowania, jak na przykład Java, posiadają wbudowaną obsługę interfejsów. Dlatego komponenty J2EE (ang. *Java 2 Enterprise Edition*) mogą być w całości pisane w jednym języku, co ułatwia kompilację kodu.

W Mozilli do tego celu służy XPIDL, który jest bardzo podobny do języka IDL[4] zdefiniowanego w standardzie CORBA (ang. *Common Object Request Broker Architecture*). Wykorzystanie odrębnego języka do projektowania interfejsów jest konieczne, aby umożliwić dostęp do komponentów z poziomu różnych języków programowania.

Interfejs kompiluje się do postaci tzw. biblioteki typów (ang. *type library*) używanej podczas działania platformy. Z powodów formalnych konieczne jest również wygenerowanie klasy *proxy*, czyli kompilacja do kodu C++. Klasa ta jest następnie dołączana podczas kompilacji kodu korzystającego z danego komponentu. Dodatkowo na podstawie interfejsu można uzyskać szkielet (ang. *stub*) implementacji C++ w postaci makra, które zawiera deklaracje metod interfejsu. Kompilator XPIDL umożliwia też generowanie klas *proxy* w języku Java oraz dokumentacji w formacie HTML [11].

Zarządzanie komponentami

Zarządzaniem komponentami zajmuje się menedżer komponentów (ang. *Component Manager*). Podczas uruchamiania platformy automatycznie rejestrowane są komponenty, których skompilowane interfejsy znajdują się w podkatalogu `components`. W repozytorium przechowywane są identyfikatory UUID (ang. *Universally Unique Identifier*) wszystkich komponentów i interfejsów.

Komponenty tworzą tzw. moduł, który jest ładowany dynamicznie do pamięci dopiero w momencie użycia (tj. wywołania metody należącej do jednego z zaimplementowanych interfejsów). Ta ważna własność daje dużą oszczędność pamięci i umożliwia szybki start platformy. Wywołanie może nastąpić z innego komponentu albo z zewnątrz za pośrednictwem warstwy XPConnect. Oba przypadki zostały przedstawione na diagramie 4.1 (kółkami oznaczono interfejsy).

[3] Inny komponent może wywoływać metody interfejsu nie znając jego implementacji. Tak jest na przykład w MSCOM, którego kod źródłowy jest niejawny.

[4] Niewielkie różnice wynikają z dodania obsługi specyficznych cech platformy, takich jak typ danych `wstring` służący do kodowania napisów w formacie UTF-16.

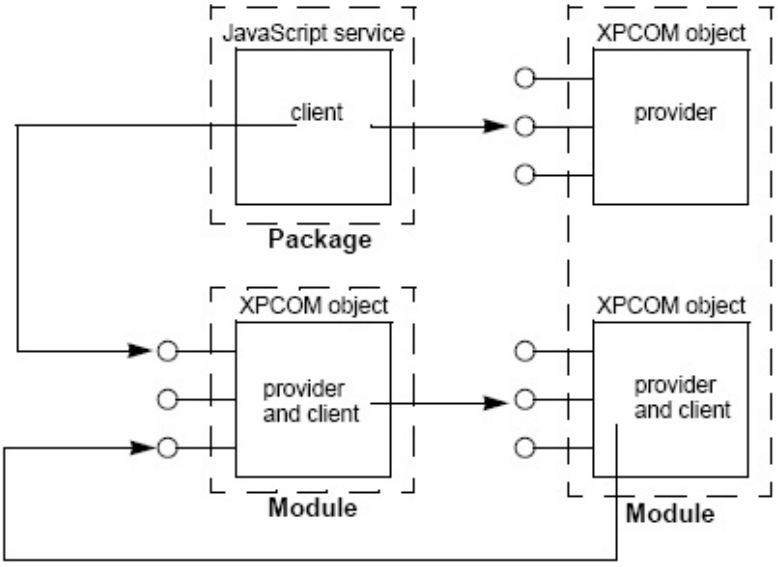


Diagram 4.1: Model XPCOM [19]

Obiekty komponentów powstają przy użyciu fabryk, ponieważ bezpośredni dostęp do implementujących klas nie jest możliwy. Standardowa fabryka może być przesłonięta przez implementację programisty. Usługi (ang. *XPCOM services*) są szczególnym rodzajem komponentów, których obiekt istnieje tylko jeden.

4.2.2. Narzędzia XPCOM

Do XPCOM oprócz komponentów należą charakterystyczne dla platformy aplikacji narzędzia ułatwiające pracę programistom.

Inteligentne wskaźniki

Programowanie w C++ wymaga ręcznego przydzielania i zwalniania pamięci, co komplikuje kod źródłowy i stwarza niebezpieczeństwo wycieków pamięci (ang. *memory leaks*). Zadanie inteligentnych wskaźników (ang. *smart pointers*) polega na automatycznym zliczaniu referencji oraz zwalnianiu pamięci. W XPCOM są to wskaźniki na interfejsy komponentów, które zawsze dziedziczą po `nsISupports`. W połączeniu z zastosowaniem wzorców C++ (klasa `nsCOMPtr`) możliwe jest przezroczyste dla programisty zarządzanie referencjami.

Inteligentne wskaźniki w XPCOM posiadają kilka dodatkowych własności. Najbardziej użyteczną jest możliwość niejawnego rzutowania komponentu na jeden z implementowanych interfejsów. Dzięki temu nie jest konieczna ręczna konwersja typów.

Operacje na ciągach znaków

Biblioteka funkcji operujących na ciągach znaków zwalnia programistę z posługiwania się wskaźnikami na typ `char`. Ma to szczególne znaczenie w Mozilli, ponieważ w formacie UTF jeden znak może zajmować do 4 bajtów. Biblioteka ta posiada wiele zaawansowanych możliwości, jak zarządzanie ciągami podzielonymi na fragmenty, konwersja

między wieloma standardami kodowania i automatyczne zwalnianie pamięci. Programista ma do dyspozycji kilkanaście specjalizacji klasy `nsString`, które różnią się między sobą wewnętrzną reprezentacją ciągów znaków, metodami zarządzania pamięcią oraz zastosowaniami [15].

Makra
Mozilla definiuje wiele makr, których stosowanie zwiększa przejrzystość kodu i przyspiesza programowanie, co jest celem każdej platformy aplikacji[5]. Makra można podzielić na trzy kategorie:

- standardowa implementacja modułu – makra rozwijane do klasy implementującej moduł złożony z podanych komponentów;
- metody komponentu – standardowa fabryka i metody `nsISupports` oraz deklaracje metod interfejsu (wygenerowane przez kompilator XPIDL);
- makra do testowania aplikacji – asercje, ostrzeżenia i komunikaty o błędach.

4.2.3. Narzędzia niskopoziomowe

Poniżej XPCOM w architekturze Mozilli znajduje się warstwa NSPR (ang. *Netscape Portable Runtime Library*) tworząca abstrakcję systemu operacyjnego. Jej celem jest zapewnianie przenośności aplikacji. NSPR implementuje następującą funkcjonalność:

- I/O – abstrakcja systemu plików oraz gniazd sieciowych;
- wątki – synchronizacja oparta na monitorach i zmiennych warunkowych;
- adresy sieciowe – IPv4 oraz IPv6;
- czas – abstrakcja systemowego zegara oraz funkcje kalendarza;
- zarządzanie pamięcią – funkcje przydzielania i zwalniania pamięci;
- dynamiczne biblioteki – ładowanie i usuwanie z pamięci na poziomie systemu operacyjnego [17].

4.3. Model danych

Zarządzanie danymi należy do zadań bardzo wielu aplikacji, dlatego ważne jest wsparcie ze strony platformy w tym zakresie. Mozilla zawiera implementację bardzo elastycznego modelu danych, opartego na RDF i zintegrowanego z interfejsem użytkownika.

4.3.1. Producent-konsument

Model zarządzania danymi w Mozilli opiera się na wzorcu projektowym producent-konsument. Producent (nazywany *data source*) dostarcza dane w formacie RDF, które są następnie przetwarzane przez konsumenta (tzw. *content sink*). Wewnątrz platformy na strumieniach RDF mogą być wykonywane operacje, na przykład łączenia i dzielenia.

Źródła danych mogą znajdować się na zdalnym komputerze. W takim przypadku dane RDF przesyła się przez sieć dokładnie tak samo, jak dokumenty HTML. Wzorzec producent-konsument zapewnia dużą elastyczność zarówno dla źródeł danych, jak i sposobu ich prezentacji.

[5] Por. definicja na stronie 16.

4.3.2. Rola RDF

RDF to model metadanych przeznaczony do opisywania zasobów oraz ich wzajemnych relacji. Ma formę grafu skierowanego, złożonego ze stwierdzeń postaci *podmiot-predykat-obiekt*. RDF jest rodziną specyfikacji W3C obejmującą zgodny z XML format dokumentu [10].

RDF jest stosowany w Mozilli m.in. do zarządzania zakładkami, wiadomościami e-mail, grupami UseNet i wynikami wyszukiwania. Ze względu na swój charakter model ten szczególnie dobrze nadaje się do danych hierarchicznych i danych o luźnej strukturze. Przykładami aplikacji mogą być systemy zarządzania treścią i wizualizacji sieci. Znacznie mniej naturalne jest zastosowanie RDF do danych „tabelkowych", chociaż z braku innej możliwości niekiedy jest konieczne.

Ponieważ RDF jest również formatem pliku, źródło danych można połączyć z określonym dokumentem, który będzie przez nie odczytywany i zapisywany. Ten mechanizm serializacji używany jest także do przesyłania danych przez sieć. Źródło danych może w całości znajdować się w pamięci albo łączyć się z bazą danych. Przykładem zaawansowanych możliwości jest leniwe obliczanie wartości w czasie działania aplikacji. Wykorzystują to dynamiczne zakładki w Firefoksie (ang. *live bookmarks*), które na żądanie otrzymują od zdalnego serwera komunikaty RSS.

4.3.3. Reprezentacja danych

Statyczna część interfejsu użytkownika opisywana jest przez dokumenty XUL, natomiast zawartość kontrolek takich jak drzewo i lista jest generowana dynamicznie na podstawie szablonów. Dane pochodzące ze wskazanego strumienia RDF są automatycznie przetwarzane i w odpowiedni sposób prezentowane. Dlatego jedno źródło danych może być przedstawiane wielokrotnie na różne sposoby: jako drzewo, hierarchiczne podmenu czy lista.

Szablony XUL otrzymują powiadomienia o zmianach w źródłach danych i uaktualniają ich reprezentację. Interakcje z użytkownikiem przebiegają w taki sam sposób jak w pozostałych kontrolkach. Jest to więc system sterowany danymi (ang. *data-driven*) [9].

4.4. Rozszerzenia

Mozilla posiada wiele własności ułatwiających rozszerzanie jej aplikacji. Podrozdział ten dotyczy zagadnień związanych z dodawaniem nowej funkcjonalności do istniejącej aplikacji (np. Firefoksa), w przeciwieństwie do rozszerzania samej platformy. WebMemo jest przykładem takiego rozszerzenia.

4.4.1. Integracja z aplikacją

Z punktu widzenia platformy komponenty dostarczane przez rozszerzenie w niczym nie różnią się od komponentów Mozilli. Menedżer komponentów automatycznie rejestruje moduł rozszerzenia, integracja niskopoziomowa nie wymaga więc dodatkowej pracy.

Aby umożliwić użytkownikowi dostęp do nowej funkcjonalności, rozszerzenie posiada zazwyczaj własny interfejs. Integracja z interfejsem aplikacji następuje za pośrednictwem tzw. nakładek (ang. *XUL Overlays*). Nakładka umożliwia modyfikację dowolnego fragmentu oryginalnego interfejsu, jak na przykład dodanie nowej pozycji w menu albo na pasku narzędzi. Dzięki tak elastycznemu mechanizmowi interfejs rozszerzenia nie wyróżnia się w stosunku do interfejsu aplikacji.

Oprócz interakcji z użytkownikiem rozszerzenie może reagować na zdarzenia wewnętrzne platformy. Mozilla definiuje wiele specjalistycznych *obserwatorów*, których metody są wywoływane w takich sytuacjach jak zakończenie ładowania strony WWW czy dodanie nowego elementu do historii. Podobnie jak są wykorzystywane przez komponenty platformy, mogą służyć do innych celów w rozszerzeniach.

4.4.2. Instalacja

Instalacja rozszerzenia w Mozilli jest prosta nie tylko dla użytkownika, ale również dla programisty. Dzięki technologii XPInstall nie ma potrzeby używania niezależnych instalatorów, takich jak *InstallShield* czy `rpm(1)`. Menedżer rozszerzeń (ang. *Extension Manager*) odczytuje plik konfiguracyjny w formacie RDF i opcjonalnie wykonuje skrypt instalacyjny JavaScript. Jedynym zadaniem programisty jest dostarczenie pliku konfiguracyjnego i samego pakietu.

Menedżer rozszerzeń posiada interfejs służący do zarządzania zainstalowanymi pakietami. Możliwe są m.in. deinstalacja i aktualizacja, konfiguracja opcji oraz odwiedzenie strony domowej rozszerzenia. Jest to wygodne dla użytkownika, a programistę zwalnia z konieczności pisania odpowiedniego kodu.

4.4.3. Bezpieczeństwo

Jak zaznaczono w podrozdziale 3.3 dla Mozilli nie istnieje wyraźna granica między interfejsem użytkownika a stroną WWW. Dlatego zdalny serwer może udostępniać strony napisane w języku XUL, które po załadowaniu staną się częściami interfejsu przeglądarki. Przykłady to *Mozilla Amazon Browser* [44] i gry on-line [45]. Są to klienckie aplikacje sieciowe o konstrukcji podobnej do DHTML i znacznie większych możliwościach.

Ścisła integracja zdalnego kodu z przeglądarką wymaga szczególnego uwzględnienia problemu bezpieczeństwa. W Mozilli zdalny kod ma zawsze ograniczone uprawnienia i nie posiada dostępu do lokalnych plików czy zakładek użytkownika. Stosowana jest polityka wspólnego pochodzenia (ang. *Same-Origin Policy*), zgodnie z którą dany kod może wykonywać ryzykowne operacje wyłącznie na zasobach pochodzących z tego samego źródła [32]. Istnieje możliwość rozluźnienia ograniczeń za zgodą użytkownika.

Adresy URL dokumentów należących do platformy (m.in. zainstalowanych rozszerzeń) posiadają prefiks `chrome://`. Taki kod nie ma ograniczonych uprawnień.

4.5. Możliwości i ograniczenia

Opisane cechy charakterystyczne Mozilli decydują o jej możliwościach i ograniczeniach. Ta nieustannie ewoluująca platforma bardzo dobrze sprawdza się w określonych zastosowaniach.

4.5.1. Ewolucja platformy

Platforma Mozilla rozwinęła się z przeglądarki internetowej, która do dziś jest jej najważniejszą aplikacją. Ten związek ma kilka ważnych konsekwencji.

Komponenty XPCOM koncentrują się na zagadnieniach sieciowych i chociaż obecnie jest ich ponad tysiąc, nie dorównują wszechstronnością bibliotekom takich platform jak Java.

Od chwili powstania Mozilla znacznie zwiększyła swój zakres zastosowań, jednak aplikacje sieciowe pozostają główną jej siłą.

Rozwój platformy jest zależny od rozwoju przeglądarki, można nawet powiedzieć, że jest jej podporządkowany. W konsekwencji Mozilla pozostaje ukierunkowana na zastosowania sieciowe, co jest jej specjalizacją, ale też ograniczeniem.

Ponadto rozwój Mozilli i Firefoksa jest efektem pracy jednej społeczności. Dlatego przeglądarka bardzo dobrze wykorzystuje możliwości platformy i jest jej wzorcowym zastosowaniem. Z powodu braku dobrej dokumentacji pozostali programiści znajdują się w znacznie trudniejszej sytuacji.

4.5.2. Konsekwencje architektury

W architekturze Mozilli warstwa prezentacji jest wyraźnie oddzielona od warstwy logiki, a struktura interfejsu użytkownika jest niezależna od jego wyglądu i funkcjonalności. Luźno powiązane komponenty komunikują się ze sobą za pośrednictwem interfejsów. Zastosowanie różnych technologii w poszczególnych częściach systemu zapewnia ich separację, także w aplikacjach platformy.

Taka architektura ma wiele zalet: łatwość utrzymywania i testowania kodu, możliwość wielokrotnego wykorzystania, skalowalność. Ponadto w modelu warstwowym modyfikacja jednej warstwy powoduje konieczność zmian w co najwyżej dwóch sąsiednich. Cechy te są bardzo ważne dla Mozilli, dużego projektu typu open-source, rozwijanego przez społeczność internetową.

Mozilla jest też oparta na modelu MVC (ang. *Model-View-Controller*) [9]. Najogólniej mówiąc, widokiem jest XUL, kontrolerem JavaScript, a modelem XPCOM. Takie spojrzenie podkreśla rolę JavaScriptu jako technologii przekształcającej zdarzenia interfejsu w akcje modelu. Rezultat jest podobny: niezależność poszczególnych składników platformy.

Zastosowanie technologii COM zapewnia Mozilli elastyczność także na poziomie logiki systemu. Implementacje komponentów mogą się dowolnie zmieniać w obrębie swoich interfejsów bez konieczności modyfikacji innych części kodu. Modularyzacja logiki znacznie ułatwia przeprowadzanie testów jednostkowych, a także rozszerzanie funkcjonalności systemu.

4.5.3. Konsekwencje technologii

Bardzo ważną zaletą Mozilli jest wspieranie i wykorzystywanie standardowych technologii. Programiści mogą stosować istniejące narzędzia oraz posiadane umiejętności posługiwania się językami typu XML czy JavaScript. Użytkownicy znający podstawy programowania są w stanie tworzyć tematy i proste rozszerzenia. Projektanci stron WWW w podobny sposób mogą programować interfejs użytkownika. Ważna jest też promocja przestrzegania standardów W3C stosowanych w Internecie.

Innowacyjny sposób konstrukcji interfejsu użytkownika w Mozilli okazał się sukcesem, o czym świadczy chociażby zaadoptowanie go w Microsoft Windows Vista [8]. Jak to już zostało stwierdzone, głównym zastosowaniem platformy jest tworzenie interaktywnych, wizualnych aplikacji. Interfejs ma budowę warstwową, gdzie dwie najbardziej zewnętrzne warstwy oznaczają wygląd i sposób działania. Ich modyfikacja nie wymaga nawet znajomości wewnętrznych warstw. Dzięki zastosowaniu technologii XUL interfejs szybko reaguje na polecenia użytkownika.

Język XML odgrywa szczególną rolę tak w Mozilli, jak i w Internecie. Usługi sieciowe wykorzystują XML jako środek komunikacji z wieloma językami programowania. Podobnie

wewnątrz Mozilli stosowany jest format RDF, w którym dane mogą być przesyłane do zdalnego serwera sieciowego[6].

Projektanci stron WWW cenią JavaScript za jego siłę wyrazu przy zachowaniu łatwości programowania oraz za szybkość (kod źródłowy jest niewielki i w całości interpretowany po stronie klienta). Największe trudności są wynikiem różnic w obsłudze JavaScriptu w przeglądarkach oraz problemów związanych z bezpieczeństwem. Z punktu widzenia Mozilli nie istnieje problem zgodności z innymi przeglądarkami, ważną zaletą jest dobra współpraca z wieloma językami programowania, a wadą – słaba wydajność w porównaniu do C++. Dużo uwagi poświęcono zapewnieniu wysokiego poziomu bezpieczeństwa.

Sam JavaScript, chociaż bardzo popularny, jest często krytykowany. Dobrym przykładem stylu programowania narzucanego przez ten język jest sposób implementacji klas i obiektów – klasę można zdefiniować na kilka sposobów, z których żaden nie przypomina tradycyjnych konstrukcji znanych z innych języków. Mimo pozornego podobieństwa do Javy, jej znajomość prowadzi częściej do nieporozumień niż do szybkiego opanowania JavaScriptu [14].

Mozilla stosuje technikę *Near Zero Validation*, która promuje zachowywanie standardów bez wymuszania go. Dokumenty XML są sprawdzane syntaktycznie, natomiast w przypadku wystąpienia błędu przetwarzanie jest kontynuowane. Podobnie błędy w JavaScript nie przerywają działania aplikacji. To zachowanie jest dobre dla użytkownika, ale niewygodne dla programisty.

NSPR zawiera implementację wątków, łącznie z mechanizmami synchronizacji. Chociaż pisanie wielowątkowych komponentów jest możliwe, większość istniejących nie posiada tej własności. Co więcej, w warstwie wysokopoziomowej wszystkie operacje przebiegają w jednym wątku. W konsekwencji cały interfejs użytkownika może zostać zablokowany przez jedną funkcję JavaScript [9].

Mozilla posiada zaawansowany mechanizm zarządzania rozszerzeniami i zapewniania bezpieczeństwa przy uruchamianiu zdalnego kodu. Jednak nie została przewidziana sytuacja, w której dana aplikacja rozproszona działa w zamkniętym i kontrolowanym środowisku. W rezultacie nie ma możliwości automatycznego zniesienia wszystkich ograniczeń, co znacznie usprawniłoby jej działanie.

4.5.4. Zastosowanie do szybkiego tworzenia aplikacji

Mozilla posiada kilka cech pozwalających na stosowanie jej zgodnie z modelem RAD (ang. *Rapid Application Development*). O ile sama platforma ewoluuje powoli w kierunku wszechstronności, o tyle na jej podstawie można szybko tworzyć konkretne rozwiązania, co jest celem RAD. Następujące własności są charakterystyczne dla takiego modelu programowania [9].

Programowanie wysokopoziomowe

W Mozilli obowiązującą zasadą jest używanie do każdego zadania właściwego narzędzia. Programowanie jest bardziej wydajne, gdy zamiast C++ wykorzystuje się JavaScript, a interfejs można opisać przy pomocy deklaratywnego XUL. Rezultatem jest znacznie szybsze dostarczanie gotowych rozwiązań.

Prototypowanie interfejsu użytkownika

W modelu RAD używa się narzędzi do projektowania interfejsu użytkownika, a następnie generowania kodu źródłowego. W Mozilli nie ma takiej potrzeby, ponieważ sam

[6] W Javie tę samą funkcję spełnia RMI (ang. *Remote Method Invocation*), gdzie kod bajtowy odpowiada XML. To rozwiązanie nie zyskało w Internecie dużej popularności, ponieważ jest ograniczone do jednego języka programowania.

dokument XUL jest prototypem interfejsu, a funkcje obsługi JavaScript mogą zostać dodane później. Ustalanie z klientem kształtu interfejsu we wczesnej fazie projektu zapewnia rozwój aplikacji we właściwym kierunku.

Specjalistyczne rozwiązania
Celem projektu jest rozwiązanie konkretnego, a nie ogólnego problemu. Mozilla jest wyspecjalizowana w obszarze zastosowań sieciowych i dostarcza gotowe komponenty do tworzenia tego typu aplikacji. Wykorzystywanie istniejącego kodu zamiast tworzenia nowego jest kolejną zasadą modelu RAD.

4.5.5. Licencjonowanie

Fundacja Mozilla opracowała dla swoich projektów licencję MPL (ang. *Mozilla Public License*) [27]. Jest ona bardzo podobna do GPL, tak że alternatywnie do MPL 1.1 może obowiązywać GPL 2.0 albo LGPL 2.1. Licencja Mozilli jest mało restrykcyjna, co jest kolejną zaletą platformy.

W sekcji 4.4.1 został przedstawiony mechanizm integracji aplikacji z platformą. Stosowanie nakładek XUL umożliwia wydzielanie kodu użytkownika od kodu Mozilli. Jest to o tyle ważne, że wymienione licencje zobowiązują do publicznego udostępniania zmodyfikowanego kodu platformy, ale nie wykorzystującej go aplikacji. W ten sposób kod źródłowy rozszerzenia może pozostać własnością jego autora.

Rozdział 5

WebMemo – rozszerzenie Firefoksa

Rozdział ten został poświęcony rozszerzeniu WebMemo, które jest projektem magisterskim autora niniejszej pracy. Celem jest nowa implementacja mechanizmu obsługi historii odwiedzanych stron WWW w Firefoksie. Przy omawianiu WebMemo przedstawiono praktyczne zastosowanie opisanych w poprzednich rozdziałach technologii.

5.1. Wprowadzenie

Firefox osiągnął ogromny sukces. Ogólnoświatowy udział rynkowy tej przeglądarki utrzymuje się na poziomie kilkunastu procent, a w Polsce przekracza 20 %[1]. Oficjalna strona Mozilli podaje 21 wyróżnień zdobytych w latach 2004 i 2005 [25]. Entuzjazm użytkowników i programistów jest widoczny również w nieustannie wzrastającej liczbie rozszerzeń.

Zgodnie z założeniem projektowym Firefox oferuje jedynie podstawową funkcjonalność. Jego twórcy skoncentrowali się na optymalizacji najważniejszych części kodu, pozostawiając rozszerzenia innym programistom. Na przykład *FlashGot* [38] implementuje menedżera pobierania plików, a *SessionSaver* [47] zarządzanie sesjami. Podobnie WebMemo rozszerza funkcjonalność przeglądarki.

A jednak sam Firefox nie jest tak dopracowany, jak mogłoby się wydawać. Użytkownik może zauważyć powolny start przeglądarki, ale większość problemów jest widoczna dopiero przy analizie kodu źródłowego. Niektóre fragmenty są w oczywisty sposób nieefektywne, co niekiedy jest nawet zaznaczane w komentarzach. Ponadto zaskakujące jest naruszanie kilku asercji podczas uruchamiania Firefoksa skompilowanego z oficjalnych źródeł stabilnej wersji 1.5. W tym zakresie wiele jest do zrobienia, jednak nie jest to łatwe przy pomocy rozszerzeń – bez modyfikacji kodu przeglądarki. W dalszej części rozdziału opisano sposób, w jaki WebMemo z powodzeniem rozwiązuje jeden z problemów dotyczący obsługi historii.

5.2. Mechanizm historii w Firefoksie

W modelu programowania Mozilli aplikacja składa się z dwóch części: interfejsu użytkownika i komponentów. Głównym zadaniem warstwy niskopoziomowej w przypadku historii Firefoksa jest obsługa bazy danych. W następującym opisie przedstawiono ograniczenia aktualnej implementacji tego mechanizmu.

[1] Źródło danych: http://ranking.pl.

5.2.1. Baza danych w Mozilli

Jedną z istotnych cech platformy aplikacji jest standard dostępu do bazy danych. W Javie za pośrednictwem JDBC można łączyć się z wieloma systemami baz danych. Sytuacja w Mozilli jest niestety znacznie trudniejsza.

Początkowo Netscape Navigator wykorzystywał zewnętrzną bibliotekę Berkeley DB, co funkcjonowało dobrze pomimo takich ograniczeń, jak brak wbudowanego języka zapytań. W wersji 4.5 próbowano zastosować obiektową bazę danych, jednak bez powodzenia. Potrzeba nowego rozwiązania pojawiła się w momencie powstania Mozilli, której licencja jest niezgodna z licencją Berkeley DB.

Wtedy zaprojektowano interfejs MDB (ang. *Message Database*) – warstwę pośrednią pomiędzy bazą danych a aplikacją. Pierwszą tymczasową implementację nazwano Mork i dołączono do darmowej Mozilli. W założeniu kolejne wersje Navigatora miały wykorzystywać inną, komercyjną implementację, która jednak nigdy nie powstała. Co więcej, Mork jest nadal j e d y n ą implementacją interfejsu MDB.

5.2.2. Struktura Mork

Mork definiuje pseudotekstowy format pliku, w którym można zapisywać dane zarówno tekstowe, jak i binarne. Wymagania były następujące:

- odporność na awarie – dane muszą pozostać spójne w przypadku przerwanej operacji zapisu;
- czytelność – format tekstowy z podziałem na wiersze;
- elastyczność – możliwość rozszerzania schematu danych w przyszłości;
- efektywność – oszczędność pamięci przez ograniczanie dodatkowych znaczników.

Powstać miał format czytelny dla człowieka i prosty w interpretacji przez komputer. Niestety nie osiągnięto ani pierwszego, ani drugiego celu. Jak niefortunnie połączono wymagania najlepiej ilustruje przykładowy kod w tym języku:

```
// <!-- <mdb:mork:z v="1.4"/> -->
< <(a=c)> // (f=iso-8859-1)
  (8A=Typed)(8B=LastPageVisited)(8C=ByteOrder)
  (80=ns:history:db:row:scope:history:all)
  (81=ns:history:db:table:kind:history)(82=URL)(83=Referrer)
  (84=LastVisitDate)(85=FirstVisitDate)(86=VisitCount)(87=Name)
  (88=Hostname)(89=Hidden)>

<(80=LE)(81=http://www.google.pl/)(82=1149525240078125)(83=google.pl)
  (84=G$00o$00o$00g$00l$00e$00)(85=http://www.mimuw.edu.pl/)(86
    =1149525244875000)(87=mimuw.edu.pl)(88=1)(89=2)(8A
    =http://www.mimuw.edu.pl/?PHPSESSID=97068a8b7280ecb348a3c4083ffee591)
  (8B=M$00I$00M$00U$00W$00 $00-$00 $00S$00t$00r$00o$00n$00a$00 $00G$00B$01$F3$00\
w$00n$00a$00)(8C=http://www.uw.edu.pl/)(8D=1149525274625000)(8E=uw.edu.pl)
  (8F=U$00n$00i$00w$00e$00r$00s$00y$00t$00e$00t$00 $00W$00a$00r$00s$00z$00a$00\
w$00s$00k$00i$00)>
1:^80 (k^81:c)(s=9)[1(^8C=LE)]
  [2(^82^81)(^84^82)(^85^82)(^88^83)(^87^84)]
  [3(^82^85)(^84^86)(^85^86)(^88^87)(^8A=1)(^86=2)]
  [4(^82^8A)(^84^86)(^85^86)(^88^87)(^87^8B)]
  [5(^82^8C)(^84^8D)(^85^8D)(^83^8A)(^88^8E)(^87^8F)]
```

Jak widać, Mork jest formatem niezrozumiałym i skomplikowanym. Zdefiniowana w kilku pierwszych wierszach tablica mieszająca powtarzających się napisów to rezultat dążenia do efektywności, którą traci się na tekstowym kodowaniu danych binarnych. Pewne znaki specjalne poprzedza się odwróconym ukośnikiem, a inne dolarem. Stosowane na wzór C++ komentarze są w pewnych miejscach fragmentami adresów URL. Nietrudno się domyślić, że analizator składniowy takiego kodu musi być bardzo złożony. Mork również nie nadaje się do czytania przez człowieka [7].

5.2.3. Dlaczego wybrano Mork

Podobnie jak Mozilla jest platformą przeznaczoną do określonych zastosowań, tak format Mork powstał w celu przechowywania książki adresowej w programie pocztowym i historii odwiedzanych stron w przeglądarce. Przy stosunkowo niewielkiej ilości danych słaba efektywność nie stanowiła problemu. Obawiano się przede wszystkim utraty danych w wyniku awarii, co mogłoby zniechęcać użytkowników, a Mork sprawdził się jako niezawodny.

Patrząc na listę wymagań, można się zastanawiać, dlaczego nie wybrano formatu XML. Powodem była wówczas niewielka popularność tego języka i dostępność narzędzi oraz niepewna przyszłość. Dzisiaj jest oczywiste, że popełniono błąd.

Mork powstał jako tymczasowa implementacja interfejsu MDB, jednak stworzenie bardziej zaawansowanej nigdy nie miało wysokiego priorytetu. Z punktu widzenia użytkownika jedyną tego konsekwencją jest ograniczenie długości historii odwiedzanych stron WWW do kilku ostatnich dni. W przeciwnym przypadku następuje bardzo duży spadek efektywności.

Ewolucja Mozilli zmierza w innym kierunku i prawdopodobnie nigdy nie powstanie nowa implementacja MDB[2]. Widać tu słabość platformy, ponieważ nie ma możliwości efektywnego zarządzania większą ilością danych. Mork jest szczególnie uciążliwy dla twórców niezależnych aplikacji, którzy mogą mieć pod tym względem szczególne wymagania.

5.2.4. Obsługa historii i zakładek

Znacznie lepiej funkcjonuje interfejs obsługujący historię. Firefox posiada panel boczny, w którym tytuły odwiedzanych stron WWW są prezentowane w postaci listy. Możliwe jest wyszukiwanie, sortowanie i usuwanie elementów. Przy historii ograniczonej do kilku ostatnich dni jest to w zupełności wystarczające.

Baza danych została zaprojektowana w sposób charakterystyczny dla Mozilli, tzn. przechowuje dodatkowe informacje przeznaczone dla potencjalnych rozszerzeń. Każdy element historii posiada tytuł, adres strony, datę ostatniej wizyty, liczbę wizyt oraz wartość `Referrer`[3]. Ostatnie pole dodano w Firefoksie 1.5 i nie jest ono wykorzystywane w przeglądarce ani dostępne z poziomu interfejsu użytkownika. Istnieje proste rozszerzenie wykorzystujące tę funkcjonalność (por. [41]), natomiast dla WebMemo jest ona niewystarczająca. Zostało to omówione w dalszej części rozdziału.

Zakładki są obsługiwane niezależnie od historii, a interfejs użytkownika jest w ich przypadku bardziej rozbudowany. Istnieje możliwość grupowania w foldery, dodawania komentarzy i słów kluczowych oraz modyfikowania własności. Nie zmienia to faktu, że cel historii i zakładek jest ten sam. Integracja obu mechanizmów w obecnym ich kształcie nie jest możliwa:

[2] Aby zwiększyć efektywność konieczna jest rezygnacja z wymagania czytelności i wprowadzenie formatu binarnego, co zostało przedstawione dalej. Internet Explorer dzięki takiemu rozwiązaniu nie ma opisywanych problemów.

[3] Zapisywany jest tu adres URL strony, z której nastąpiło pierwsze otwarcie danej.

interfejsy znacznie różnią się funkcjonalnością, a baza danych jest zbyt nieefektywna.

Niniejsza praca dotyczy tzw. globalnej historii, która jest zapisywana na dysku. W Firefoksie istnieją ponadto historie sesji, niezależne dla każdego otwartego okna i panelu. Adresy stron WWW tworzą w nich listy posortowane zgodnie z kolejnością przeglądania. Gdy użytkownik wraca do poprzedniej strony, na tej podstawie ustalany jest jej adres URL.

5.3. Potrzeba i metody wizualizacji historii

5.3.1. Motywacja

> *Ci, którzy nie pamiętają przeszłości, są skazani na jej powtarzanie.*
> — George Santayana[4]

Przeprowadzone badania pozwalają określić najczęściej używane metody zapamiętywania informacji znalezionych w Internecie [6]. W zależności od grupy użytkowników następujące techniki są stosowane w różnym stopniu:

- wysłanie wiadomości e-mail z adresem URL danej strony do samego siebie, a w przyszłości wyszukanie jej przy pomocy programu pocztowego;
- wykorzystanie archiwum korespondencji ze współpracownikami (w wiadomościach często pojawiają się adresy URL);
- wydrukowanie strony WWW z okna przeglądarki;
- zapisanie strony na dysku;
- wklejenie adresu URL do pliku albo zewnętrznego systemu zarządzania informacją;
- dodanie dowiązania z własnej strony domowej;
- użycie wyszukiwarki internetowej (ponowne znalezienie strony);
- wpisanie adresu URL z pamięci, być może przy pomocy podpowiedzi przeglądarki;
- założenie zakładki.

Ponadto na podstawie wyników badań stwierdzono, że najrzadziej stosowanymi są mechanizmy historii i zakładek, a więc te wspierane przez przeglądarki.

Trudność w przypadku zakładek polega na konieczności ręcznego zarządzania. Statystycznie ich liczba rośnie liniowo i osiąga 200-300 po dwóch latach. Dopiero wtedy 44 % użytkowników jest skłonnych pogrupować zakładki w foldery, a 33 % nie czyni tego nigdy. Ponadto dla 86 % badanych nazwy zakładek (a więc tytuły stron WWW) często są niewłaściwe, przy czym bardzo rzadko domyślne nazwy były zmieniane.

Jak widać potrzebny jest bardziej zautomatyzowany mechanizm. Historia odwiedzonych stron WWW nie spełnia tego zadania, a jej użyteczność jest oceniana poniżej 1 %[5]. Może się to zmienić, gdy zostanie wprowadzona zaawansowana wizualizacja historii.

[4] Por. *The Life of Reason: Reason in Common Sense* (1905), cyt. za [5].
[5] W publikacji [6] podano kilka badań, w których uzyskano ten wynik.

5.3.2. Struktura WWW

Wizualizacja struktury WWW polega na graficznym przedstawieniu powiązań pomiędzy stronami internetowymi. Dowiązania pełnią rolę krawędzi, a strony wierzchołków grafu. Istnieje wiele sposobów wizualizacji, m.in. z wykorzystaniem ikon albo miniatur stron. Publikacja [2] zawiera zestawienie bardziej zaawansowanych metod operujących w przestrzeni trójwymiarowej.

Istnieją dwa rodzaje wizualizacji. Pierwszy polega na śledzeniu aktywności użytkownika i jest określany jako *bierny*. Graf tworzą jedynie strony otwierane w oknach przeglądarki, a jego głównym celem jest ułatwianie powrotu do stron wcześniejszych. W konsekwencji struktura WWW jest widziana w sposób spersonalizowany i subiektywny dla danego użytkownika [5].

Celem *aktywnego* rodzaju wizualizacji jest tworzenie obiektywnej mapy pomagającej w poruszaniu się po strukturze WWW analogicznie do mapy drogowej. Polega to na automatycznej eksploracji sąsiednich stron, dzięki czemu użytkownik może przewidzieć najlepszą ścieżkę do szukanej. Kosztem jest znacznie większe zużycie zasobów.

5.3.3. Wizualizacja w Firefoksie

Wizualizacja historii w Firefoksie sprowadza się do listy odwiedzonych stron, której długość z powodu nieefektywnej bazy danych jest ograniczona do kilku ostatnich dni (standardowo 9). W takiej sytuacji implementacja zaawansowanego mechanizmu wizualizacji nie miałaby sensu.

Historia w przeglądarce internetowej powinna działać w sposób zbliżony do ludzkiej pamięci, a więc koncentrować się bardziej na powiązaniach między stronami niż na ich tytułach. Można mówić o dwóch rodzajach biernej wizualizacji. Pierwszy dotyczy historii sesji, która nie wykorzystuje bazy danych. Z tego powodu jest prostsza w implementacji, ale też mniej przydatna. Drugi polega na wizualizacji zapisywanej na dysku globalnej historii. Jest to rozwiązanie zrealizowane w WebMemo.

5.4. Cel projektu i wymagania

Celem projektu WebMemo jest nowa implementacja globalnej historii jako rozszerzenia przeglądarki Firefox. Następujące wymagania odpowiadają opisanym wadom i brakom obecnego rozwiązania.

Efektywna baza danych
: Warunkiem efektywności systemu obsługi historii jest zastosowanie profesjonalnej bazy danych. Zamiast rozwiązania opartego na interfejsie Mork i MDB należy zaadoptować zewnętrzną bazę danych obsługującą język SQL. Wymagana jest przede wszystkim efektywność i odporność na awarie, przy czym dane nie muszą być zapisywane w formacie tekstowym.

Wizualizacja historii
: WebMemo ma implementować interfejs użytkownika przedstawiający historię w postaci grafu. Celem wizualizacji historii jest znaczne zwiększenie użyteczności tego składnika przeglądarki.

Zakładki
: Dodatkowo ma istnieć możliwość tworzenia zakładek z elementów historii. Ma to być nie tyle pełna obsługa zakładek, co demonstracja korzyści wynikających z jej integracji z mechanizmem historii. Na poziomie bazy danych historia i zakładki mają używać tych samych tabel.

Niezależność od obecnej implementacji
: WebMemo ma być niezależne od istniejącego systemu obsługi historii, a w konsekwencji wymagać jedynie nieznacznych modyfikacji dla kolejnych wersji Firefoksa. Należy zapewnić możliwość jednoczesnego używania obu implementacji historii, a deinstalacja rozszerzenia ma nie powodować utraty danych w formacie Mork.

Prosta instalacja
: Zastosowanie technologii XPInstall gwarantuje prostą instalację rozszerzenia, o ile nie ma dodatkowych wymagań. WebMemo nie może zakładać, że system operacyjny posiada zainstalowaną bazę danych, taką jak PostgreSQL. Nie może też modyfikować kodu samego Firefoksa.

Przenośność i lokalizacja
: Rozszerzenie ma poprawnie wykorzystywać możliwości platformy, tzn. kod C++ ma być przenośny, a tłumaczenia powinny ograniczać się do edycji plików DTD.

Implementacja WebMemo wymaga wysokiego stopnia integracji z platformą i wykorzystania oferowanych przez nią możliwości. Dlatego rozszerzenie to będzie bardzo dobrym przykładem zastosowania Mozilli do tworzenia zaawansowanych aplikacji.

5.5. Sposób realizacji wymagań

Jednym z postawionych wymagań jest niezależne funkcjonowanie nowej implementacji od obecnej. Oczywiście nie wyklucza to wykorzystywania w WebMemo pewnych składników istniejącego mechanizmu. Wydaje się, że oba rozwiązania mają bardzo wiele cech wspólnych.

Historia reprezentowana jest przez elastyczne źródło danych, któremu w interfejsie użytkownika może odpowiadać tak lista, jak i graf. Zatem wystarczyłoby wymienić bazę danych i połączyć ją z istniejącym strumieniem RDF. Niestety takie rozwiązanie nie jest możliwe z dwóch powodów.

Po pierwsze, konieczna jest modyfikacja samego modelu danych. Pole `Referrer` jest niewystarczające, ponieważ przechowuje tylko jeden adres URL – strony, z której nastąpiło p i e r w s z e otwarcie. Do wizualizacji w postaci grafu niezbędne są dodatkowe informacje. Z kolei zmiana modelu pociąga za sobą zmianę źródła danych.

Po drugie, mechanizm rozszerzeń Mozilli jest przeznaczony do dodawania nowej funkcjonalności, a nie modyfikowania istniejącej. Dlatego zastąpienia systemu bazy danych przy pozostawieniu kodu jej obsługi nie da się przeprowadzić bez ingerencji w samą platformę. Niemożliwa jest też modyfikacja źródła danych historii, można co najwyżej zastąpić je innym. Jeżeli WebMemo ma pozostać rozszerzeniem, to musi dostosować się do tych ograniczeń.

W rezultacie WebMemo jest całkowicie niezależne od istniejącej implementacji i nie wykorzystuje żadnego z jej elementów. We własnym zakresie przechowuje dane, które pozwalają na odtwarzanie grafu historii. Ze względu na brak kontrolki XUL reprezentującej graf zrealizowano wizualizację w postaci drzewa.

5.5.1. SQLite

Zgodnie z wymaganiami baza danych ma być stabilna, efektywna, obsługująca język SQL i niewymagająca instalacji ani konfiguracji. Dobrym rozwiązaniem jest niewielka, zagnieżdżana biblioteka SQLite, która implementuje większą część standardu SQL92 oraz transakcje

spełniające warunki ACID. Baza danych jest przechowywana w jednym pliku, którego rozmiar zwiększa się proporcjonalnie do ilości informacji. SQLite można dowiązać statycznie do aplikacji C/C++, wtedy nie jest konieczna instalacja.

Wybór opisanej bazy danych został dokonany przez projektantów Mozilli, czego dowodem jest dostarczanie kodu źródłowego SQLite wraz z platformą[6]. Chociaż nie wykorzystuje się go w obecnej wersji Mozilli, w kolejnej ma zastąpić nieefektywną bazę danych Mork. WebMemo już teraz wykorzystuje duże możliwości SQLite.

5.6. Architektura WebMemo

W opisie architektury WebMemo szczególny nacisk położono na integrację z Firefoksem. Rozszerzenie wykorzystuje wszystkie elementy platformy opisane w poprzednim rozdziale.

5.6.1. Logiczna dekompozycja

Rozszerzenie WebMemo składa się z dwóch części. Pierwsza implementuje interfejs użytkownika i została napisana wyłącznie przy użyciu języków skryptowych. Drugą tworzą komponenty XPCOM, z których najważniejsze są źródła danych i baza danych. Wybranym aspektom programowania niskopoziomowego został poświęcony kolejny podrozdział. Diagram 5.1 przedstawia schemat logicznej dekompozycji WebMemo.

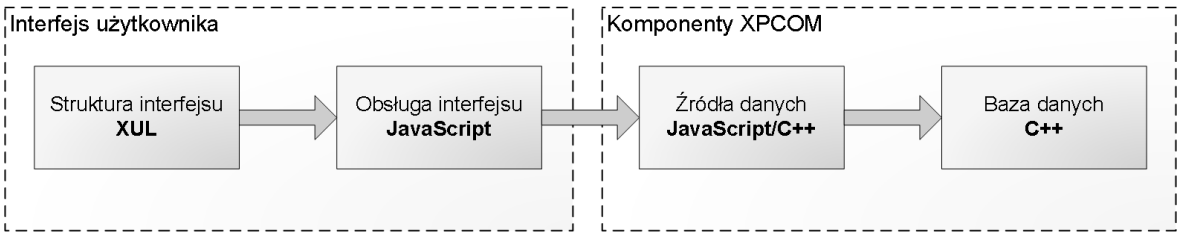


Diagram 5.1: Architektura WebMemo

Komunikacja pomiędzy elementami architektury przebiega przede wszystkim w sposób pokazany na diagramie, przy czym należy pamiętać, że jest to uproszczenie. Z punktu widzenia XPCOM wszystkie komponenty są równorzędne, a więc JavaScript obsługujący interfejs może odwoływać się bezpośrednio do bazy danych.

Ciekawe, że na diagramie 5.1 jednocześnie przedstawiony jest mechanizm globalnej historii Firefoksa. To podobieństwo jest konsekwencją wspólnego modelu programowania, stosowanego w całej platformie i jej aplikacjach. Dzięki temu komunikacja pomiędzy dwoma systemami obsługi historii nie różni się od komunikacji w obrębie każdego z nich. Na przykład dla źródła danych WebMemo nie jest istotne, do kogo należą jego obserwatorzy (por. 5.6.5).

5.6.2. Interfejs użytkownika

Statyczna część struktury interfejsu, jak na przykład okno dialogowe *Właściwości elementu historii*[7], jest w całości zdefiniowana w języku XUL. Z akcjami takimi jak otwarcie i zamknięcie okna powiązane są odpowiednie funkcje obsługi. Dynamiczna część interfejsu, m.in. drzewo

[6] Zgodnie z licencją SQLite jest własnością publiczną (ang. *public domain*).

[7] Przykładowe zrzuty ekranu prezentujące interfejs użytkownika znajdują się w dodatku B.

historii, sprowadza się do użycia szablonów XUL, które komunikują się bezpośrednio z komponentami implementującymi źródła danych. Dlatego stworzenie wizualizacji historii sprowadza się do zaprogramowania odpowiedniego komponentu [3].

Do komunikacji z Firefoksem wykorzystywani są obserwatorzy oczekujący na określone zdarzenia. Na przykład za ich pośrednictwem zakończenie ładowania strony WWW powoduje wywołanie metody komponentu bazy danych WebMemo. Do integracji z interfejsem przeglądarki służą nakładki XUL.

5.6.3. Źródła danych

WebMemo posiada dwa źródła danych, których celem jest dostarczanie informacji szablonom XUL historii i zakładek. Dane wejściowe i wyjściowe są przekazywane w formacie RDF interpretowanym przez te szablony.

Zgodność ze stosowanym w Mozilli modelem danych wymaga implementacji ogólnego i złożonego mechanizmu, nawet jeśli będzie wykorzystywany w niewielkim zakresie. Interfejs `nsIRDFDataSource` deklaruje 21 metod, takich jak zapytanie o listę podmiotów przy danym obiekcie i predykacie. Zatem konieczna jest dobra znajomość RDF.

Mozilla definiuje słownictwo i nadaje poszczególnym stwierdzeniom w języku RDF określone znaczenia. Ich zrozumienie jest kluczowe dla komunikacji z Firefoksem, niestety poważną trudność stanowi brak dokumentacji[8]. Jedynym rozwiązaniem pozostaje analiza kodu źródłowego platformy. WebMemo wykorzystuje słownictwo Mozilli zgodnie z przeznaczeniem, dodaje również własne predykaty.

Zarządzanie kolekcjami elementów RDF przebiega przy pomocy enumeratorów. Istnieje interfejs służący do tego celu, wykorzystywany między innymi przez `nsIRDFDataSource`. Ponieważ jego implementacja dostosowana jest do bazy danych Mork, WebMemo definiuje własną. Kolejne wartości enumeratorów są obliczane w sposób leniwy.

Podobnie jak w przypadku globalnej historii źródła danych WebMemo pozwalają na instalowanie obserwatorów. W ten sposób Firefox otrzymuje komunikaty o zmianach w bazie danych i odpowiednio uaktualnia interfejs użytkownika. Także inne rozszerzenia, zainteresowane śledzeniem zmian w historii WebMemo, mogą wykorzystywać ten mechanizm.

Model RDF nie jest optymalny dla takiego typu danych, jaki jest związany z historią. Zamiana wierszy tabeli na trójczłonowe stwierdzenia komplikuje implementację, ale jest nieunikniona. Jedną z korzyści jest możliwość ponownego wykorzystania: drzewo w panelu bocznym i w szczegółowym widoku historii są obsługiwane przez jedno źródło danych (rys. B.1 i B.5).

5.6.4. Baza danych

Źródło danych globalnej historii w Firefoksie zawiera również kod dostępu do bazy danych Mork. Takie rozwiązanie utrudnia wymianę systemu bazy danych, a dodatkowo czyni kod nieczytelnym i skomplikowanym.

W WebMemo obsługa bazy danych SQLite została przeniesiona do oddzielnych komponentów. Wydzielenie tego fragmentu kodu jest też uzasadnione jego odrębnością od Mozilli (zależność od zewnętrznej biblioteki i związany z tym nieco inny styl programowania). Dzięki komunikacji przez interfejsy nie tylko źródła danych, ale również kod JavaScript posiada dostęp do tych komponentów.

[8] Wewnętrzne składniki platformy są bardzo słabo udokumentowane, szczególnie w porównaniu do bibliotek przeznaczonych dla twórców rozszerzeń.

Model danych składa się z dwóch tabel. Pierwsza przechowuje elementy historii i posiada następujące pola: identyfikator[9], adres URL, tytuł strony, liczba wizyt, data ostatniej wizyty, nazwa zakładki, notatki oraz flagi (czy dany element został wpisany przez użytkownika i czy jest zakładką). Druga to pary identyfikatorów oznaczające przejścia pomiędzy stronami WWW.

Zgodnie z wymaganiami zakładki nie wymagają osobnych tabel i współdzielą swoje dane z modelem historii. Komponenty obsługujące zakładki i historię również korzystają ze wspólnego kodu. Jest oczywiste, że oba mechanizmy powinny być implementowane razem, co jednak nie zostało zrealizowane w Firefoksie.

5.6.5. Przykładowy przepływ sterowania

Na diagramie 5.2 przedstawiono przepływ sterowania przy dodawaniu strony WWW do historii. Ilustruje on współdziałanie poszczególnych elementów WebMemo oraz ich komunikację z Firefoksem. Zaznaczono też podział na części odpowiadające składnikom architektury. Komponenty należące do Firefoksa (`nsTreeBoxObject`, `nsXULTreeBuilder` i `nsDocumentViewer`) tworzą interfejs użytkownika. Skryptowy obserwator `gHistoryObserver` oraz komponent C++ obsługujący bazę danych `wmHistoryDatabase` są składnikami WebMemo.

W momencie zakończenia ładowania strony WWW `nsDocumentViewer` wysyła komunikat PAGESHOW, który powoduje wywołanie zainstalowanej funkcji obsługi `PageLoad()`. Stąd przekazuje się sterowanie do komponentu `wmHistoryDatabase`, który o dodaniu nowego elementu do historii powiadamia źródło danych. Skryptowy `gHistoryObserver` rozwija drzewo w odpowiednim węźle, w którym `nsXULTreeBuilder` dodaje nowy element. Ostatnią akcją jest odświeżenie widoku.

Ten model programowania jest charakterystyczny dla aplikacji Mozilli. Dzięki wykorzystaniu istniejących komponentów kod źródłowy rozszerzenia jest niewielki, ale ściśle powiązany z platformą (co wymaga dobrej znajomości zasad jej funkcjonowania).

5.7. Opis implementacji

W praktyce programistycznej wiele trudności stwarzają szczegóły techniczne, szczególnie przy braku doświadczenia w danym zakresie. Tak jest też w przypadku Mozilli, stosunkowo mało popularnej platformy aplikacji, a wymagającej znajomości kilku języków programowania i wielu specyficznych technologii. Najprostszym sposobem uniknięcia problemów jest ograniczenie się do wysokopoziomowego JavaScriptu, co sprawdza się w niewielkich rozszerzeniach. WebMemo zostało zaimplementowane głównie w C++ zgodnie ze stylem programowania Mozilli. W tym podrozdziale znajduje się opis kilku wybranych zagadnień i napotkanych trudności.

5.7.1. Inicjowanie WebMemo

Kod inicjujący rozszerzenie może być wywołany na kilka sposobów, z których dwa zostały wykorzystane w WebMemo.

Pierwszy polega na dodaniu atrybutu `onload` do wybranej kontrolki XUL, na przykład panelu bocznego WebMemo. Niestety nie ma gwarancji, że dany element interfejsu będzie aktywny przy starcie Firefoksa – panel może być zamknięty. Dlatego lepszym rozwiązaniem jest wywołanie funkcji inicjującej podczas ładowania głównego okna przeglądarki, co jest

[9] W historii Firefoksa rolę identyfikatorów pełnią adresy URL, co przy częstym posługiwaniu się nimi jest bardzo nieefektywne.

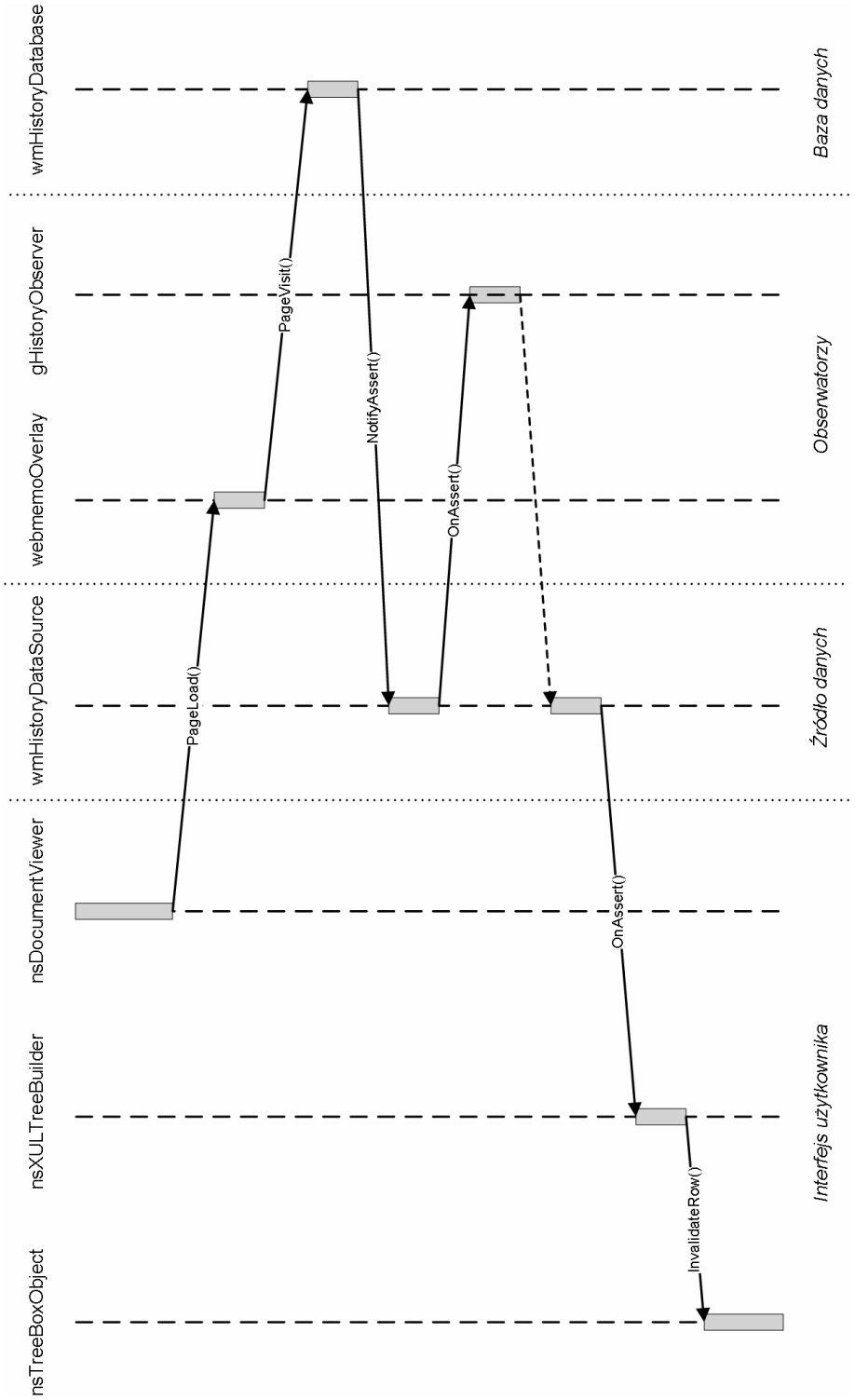


Diagram 5.2: Schemat operacji dodawania strony WWW do historii

możliwe dzięki nakładce XUL. W ten sposób WebMemo instaluje obserwatora komponentu `nsDocumentViewer`.

Drugi sposób dotyczy inicjacji przez XPCOM. Menedżer komponentów wywołuje metody inicjujące wszystkich modułów, w tym także `WebMemoModule`. Jednak następuje to przed załadowaniem profilu[10], a więc zbyt wcześnie, by można było nawiązać połączenie z bazą danych.

Mozilla stosuje wzorzec projektowy publikacji-subskrypcji (ang. *publish-subscribe*) do wysyłania powiadomień o określonych zdarzeniach. WebMemo w funkcji inicjującej moduł instaluje obserwatora, który odbiera komunikat APP-STARTUP. Jest to właściwy moment do inicjacji rozszerzenia. Otwierany jest plik bazy danych, który podobnie jak pliki historii i zakładek Firefoksa znajduje się w katalogu profilu. W analogiczny sposób komunikat QUIT-APPLICATION powoduje zakończenie działania.

5.7.2. Komponenty

Ponieważ Mozilla nieustannie ewoluuje, komponenty i interfejsy mogą w przyszłości ulegać zmianom. Ponieważ funkcjonalność zamrożonych komponentów jest prawie zawsze niewystarczająca, wydzielono najczęściej używaną część XPCOM, tzw. *XPCOM Glue*. Należą do niej niezamrożone, ale bardzo stabilne komponenty, takie jak inteligentne wskaźniki i klasy służące do operacji na ciągach znaków. Aby zachować zgodność z przyszłymi wersjami Firefoksa rozszerzenia powinny ograniczać się do tej funkcjonalności.

WebMemo w znacznie większym stopniu integruje się z platformą i z tego powodu wymaga pełnego dostępu do komponentów XPCOM. Jest to możliwe dzięki dyrektywie kompilatora `MOZILLA_INTERNAL_API`, jednak wtedy proces kompilacji staje się bardziej złożony[11].

WebMemo zawiera także własne komponenty. Interesujące jest porównanie implementujących ten sam interfejs źródeł danych `wmHistoryDataSource` i `wmBookmarksDataSource`. Pierwsze zostało napisane w C++ ze względu na większą złożoność i wymaganie efektywności. Drugie w JavaScript, ponieważ jest znacznie prostsze. Język skryptowy sprawdził się jako narzędzie do szybkiego pisania kodu, z drugiej strony w C++ można było sprawniej wykrywać błędy.

5.7.3. Styl programowania

Kod C++ Mozilli wygląda bardzo charakterystycznie z powodu pewnych konwencji oraz dużej liczby makr. Każdy identyfikator posiada określony prefiks, na przykład nazwa klasy zaczyna się od `ns` (Netscape) albo `moz` (Mozilla) [9]. W miejsce wbudowanych typów danych stosuje się odpowiednie makra NSPR (np. `PRBool`, `PRInt32`, `PRUnichar`). Słowa kluczowe, które mogą nie być dostępne w pewnych kompilatorach, również zastępuje się makrami (np. `NS_STATIC_CAST`). Metody komponentów przekazują typ `NS_RESULT`, którego wartość jest sprawdzana przez odpowiednie asercje. Zasady pisania przenośnego kodu w Mozilli zostały przedstawione w [13].

WebMemo przejmuje od Mozilli styl programowania oraz konwencje programistyczne, dzięki czemu kod źródłowy jest czytelny i przenośny. Rozszerzenie zostało przetestowane w systemach Linux i Windows.

[10] Profile odpowiadają użytkownikom w systemie operacyjnym, przy czym tutaj abstrahuje się od konkretnej platformy systemowej.

[11] Opis kompilacji WebMemo znajduje się w dodatku A.

5.7.4. Trudności techniczne

Implementacja WebMemo wymagała rozwiązania wielu trudności technicznych, z których część nie jest udokumentowana. Ponieważ doświadczenie autora może być przydatne także dla innych programistów, w tej sekcji przedstawiono kilka wybranych problemów.

Konwersja typów danych

W językach programowania odmiennie definiuje się typy danych, a do wzajemnej wymiany informacji najczęściej używa się formatu XML. Podobnie w Mozilli RDF pośredniczy między językami C++ i JavaScript. Przypuśćmy, że tytuł strony WWW ma być wypisany w oknie dialogowym *Właściwości elementu historii*. Baza danych SQLite przekazuje wskaźnik na typ `char`, któremu w Mozilli odpowiada wskaźnik na `PRChar`. Następnie tworzony jest odpowiedni podtyp `nsString`, przekształcany dalej na RDF. Skrypt otrzymuje dane w tym formacie, który konwertuje na własny typ łańcuchowy. Złożoność tej operacji wynika z natury Mozilli.

Ograniczenie XPIDL

Z konwersją typów danych związany jest jeszcze jeden problem. Ponieważ może istnieć co najwyżej jedno połączenie z bazą danych SQLite, potrzebna jest metoda do przekazywania go innym komponentom. Niestety w języku XPIDL nie przewidziano odpowiednika typu `sqlite3` ani nie zdefiniowano sposobu jego konwersji na RDF. W WebMemo zastosowano rozwiązanie polegające na bezpośrednim wywołaniu statycznej metody (z pominięciem interfejsu), a komponenty obsługujące bazę danych napisano w C++ (konwersja nie jest konieczna).

Dziedziczenie w XPCOM

W XPCOM istnieje pewien poważny, a jednocześnie subtelny problem, który w praktyce uniemożliwia wzajemne dziedziczenie komponentów. Na przykład `wmBookmarksDatabase` jest specjalizacją `wmCommonDatabase`, która w swoim interfejsie `wmICommonDatabase` posiada metodę `PrecompileQueries()`. Jeśli podklasa nie chce przesłonić tej metody, to nie może jej dodać do `wmIBookmarksDatabase` (ani odziedziczyć po `wmICommonDatabase`), ponieważ wtedy makro `NS_DECL_WMIBOOKMARKSDATABASE` zawierałoby jej deklarację. Zatem metody nieprzesłonięte znajdą się w `wmICommonDatabase`, a przesłonięte w `wmIBookmarksDatabase`. Wtedy klient nie będzie mógł zrzutować komponentu na najbardziej wyspecjalizowany interfejs, ponieważ nie znajdzie w nim wszystkich metod.

Oczywiście podział metod na interfejsy powinien wynikać z projektu, a nie implementacji. Dlatego przyjętym w WebMemo rozwiązaniem jest rezygnacja z `wmCommonDatabase` jako komponentu. Klasa ta jest dziedziczona przez komponent `wmBookmarksDatabase`, a interfejs `wmIBookmarksDatabase` rozszerza `wmICommonDatabase`.

Zarządzanie pamięcią

Nie da się całkowicie wyeliminować konieczności ręcznego zarządzania pamięcią w C++. Nawet przy zastosowaniu inteligentnych wskaźników pozostaje pewna trudność, a mianowicie podział referencji na silne i słabe. Ponadto wskaźniki te operują wyłącznie na interfejsach XPCOM, więc na przykład pamięć używana przez bazę danych SQLite musi być obsługiwana ręcznie.

JavaScript w Mozilli implementuje mechanizm odśmiecania typu *mark and sweep* [1]. Destruktory obiektów C++, dostępnych przez XPConnect, są wywoływane z pewnym opóźnieniem. Zanim to nastąpi, kod C++ może stworzyć nowy obiekt danego typu, co jest niebezpieczne chociażby w przypadku enumeratorów. Programując należy o tym

pamiętać. Ponieważ JavaScript nie zlicza referencji, inteligentne wskaźniki nie przechodzą przez interfejsy XPIDL. Przed przekazaniem do JavaScript wskaźnika komponentu XPCOM konieczne jest ręczne zwiększenie licznika referencji (`NS_ADDREF`).

5.8. Możliwości i ograniczenia

WebMemo spełnia wszystkie opisane wymagania. SQLite zapewnia efektywną bazę danych, a do wizualizacji historii służy drzewo XUL. Prosta instalacja i lokalizacja[12] oraz przenośność wynikają z poprawnego wykorzystania odpowiednich mechanizmów Mozilli. Niestety razem z możliwościami WebMemo przejmuje pewne ograniczenia platformy.

Firefox dodaje do globalnej historii adresy wszystkich stron, które są otwierane w przeglądarce, co nie zawsze jest pożądane. Na przykład mimo że `http://google.pl` jest przekierowaniem do `http://www.google.pl`, oba adresy znajdą się w historii. Adresy stron, które nie istnieją (błąd HTTP 404), są również dodawane. W innych przypadkach w globalnej historii brakuje części adresów (szczególnie przy stronach z ramkami).

Dokładnie te same strony są dodawane do historii WebMemo. Chociaż lepsze byłoby bardziej zaawansowane kryterium, problem ten jest w ogólności nierozwiązywalny. Przy pomocy zagnieżdżonego na stronie JavaScriptu można skonstruować przekierowanie niemożliwe do wykrycia. Ponadto użytkownik może chcieć przeszukiwać historię po adresie strony, która nie została dodana z powodu przekierowania. Zatem nie jest jasne, które strony należy zapisywać w historii [21].

Trudności w wizualizacji występują w przypadku serwerów, które do adresów swoich stron dodają identyfikatory sesji HTTP (np. `http://www.mimuw.edu.pl`). Ponieważ identyfikator ten jest za każdym razem inny, powtarzająca się strona będzie nieustannie dodawana jako nowa. W konsekwencji wizualizacja historii nie zadziała zgodnie z oczekiwaniem.

Ponadto WebMemo może mieć ograniczoną efektywność, jeśli katalog domowy użytkownika jest podłączony przez NFS. SQLite jako mechanizmu synchronizacji używa blokad zakładanych na plik bazy danych. W przypadku sieciowego systemu plików konieczność posługiwania się blokadami przy każdej operacji odczytu i zapisu może znacznie spowolnić aplikację. Bardziej efektywna byłaby synchronizacja przez bibliotekę SQLite [28].

5.9. WebMemo a inne rozwiązania

5.9.1. Podobne rozszerzenia

Firefox posiada bardzo wiele rozszerzeń, jednak możliwości większości z nich ograniczają się do efektów wizualnych. Również te dotyczące historii przeważnie koncentrują się na interfejsie użytkownika. Na przykład *History Submenus* [40] dodaje podmenu z globalną historią, ale nie zmienia sposobu jej działania. *How'd I Get Here* [41] wykorzystuje nową funkcjonalność historii – pokazuje wartości pola `Referrer`.

Bardziej rozbudowanym rozszerzeniem jest *Enhanced History Manager* [36]. Chociaż nie modyfikuje globalnej historii, implementuje nową jej obsługę w JavaScript. Dodaje m.in. okno menedżera historii, klawisze skrótów oraz menu kontekstowe. Elementy historii nadal są prezentowane w postaci listy – jest to tylko rozszerzenie funkcjonalności interfejsu użytkownika.

Żadne z dostępnych rozszerzeń nie używa innego niż Mork systemu bazy danych ani modelu historii. Wymagałoby to znacznie większego nakładu pracy i ścisłej integracji z Firefoksem. Dlatego programiści próbują wykorzystywać inne sposoby zapamiętywania historii.

[12] WebMemo jest standardowo dostępne w angielskiej i polskiej wersji językowej.

ForwardFork [39] używa zwykłą tablicę JavaScript do przechowywania odgałęzień historii sesji. Rozmiar danych jest niewielki, nie trzeba ich zapisywać na dysku, dlatego rozwiązanie funkcjonuje dobrze. W przypadku globalnej historii ta metoda byłaby niewystarczająca.

Istnieje jedno rozszerzenie, które implementuje wizualizację podobną do WebMemo. Napisany w JavaScript *Referrer History* [46] łączy elementy globalnej historii przez wartości `Referrer`. Ograniczeniem jest to, że każda strona może mieć tylko jedną wartość tego pola, związaną z jej pierwszym otwarciem. Podobnie jak pozostałe, rozszerzenie to używa nieefektywnej bazy danych Mork.

Chociaż architektura Mozilli pozwala na programowanie w C++, prawie wszystkie rozszerzenia wykorzystują jedynie JavaScript. Języki niższego poziomu są stosowane w większych aplikacjach platformy, natomiast bardzo rzadko w rozszerzeniach. Przewaga WebMemo polega na efektywności i sile wyrazu, jakie daje C++. Autor *Referrer History* przyznaje w opisie swojego rozszerzenia, że nadaje się ono do celów demonstracyjnych, a nie do praktycznego użytku.

5.9.2. Plany rozwojowe Firefoksa

Jedyną konkurencją dla WebMemo jest sam Firefox. Aktualna wersja 1.5 nie posiada ani efektywnej bazy danych, ani wizualizacji historii, jednak w przyszłości ma się to zmienić.

Interfejs Storage

Ewolucja Mozilli wykształciła niezależną obsługę wielu formatów danych: db1.85, RDF, XML, Mork oraz różnych formatów tekstowych. Kolejnym krokiem ma być wprowadzenie ujednoliconego dostępu do informacji, opartego na bazie danych SQLite.

Firefox 2 będzie implementował interfejs *Storage*, który jest obecnie rozwijany. Pojawią się też nowe problemy, takie jak powiadamianie obserwatorów o zmianach w bazie danych[13]. Możliwy będzie bezpośredni dostęp do tabel, jednak RDF pozostanie ze względu na zgodność z istniejącymi komponentami [22].

Storage jako nowy element platformy będzie miał przez długi czas niestabilny interfejs, z czym liczyć się muszą twórcy rozszerzeń. Jego zamrożenie jest planowane dopiero na trzecią wersję Firefoksa.

WebMemo wykorzystuje tę samą bazę danych do rozwiązania konkretnego problemu historii. Nie podejmuje próby integracji wymienionych formatów, ale ogranicza się do zastąpienia jednego. Od Storage różni się stopniem ogólności i w konsekwencji architekturą.

System Places

Places będzie nową implementacją globalnej historii wykorzystującą możliwości Storage. Dodatkowa funkcjonalność obejmie zaawansowane wyszukiwanie przy pomocy indeksu i słów kluczowych oraz dynamicznie generowane foldery z wynikami złożonych zapytań. Lista historii będzie mogła być dzielona na sesje posortowane według kolejności oglądania stron. Planowana jest też większa integracja z systemem zakładek.

Dla programistów największą zmianą będzie nowy model bazy danych. Funkcję pola `Referrer` będzie spełniać dodatkowa tabela przechowująca wszystkie przejścia pomiędzy stronami w historii. Te dodatkowe informacje mają stać się motywacją dla twórców rozszerzeń, a jedną z propozycji jest właśnie graf historii [20].

[13] Wyzwalacze w SQLite działają tylko w obrębie danej aplikacji, więc konieczne będzie wprowadzenie mechanizmu IPC.

Wbrew początkowym planom i oczekiwaniom programistów Firefox 2 nie będzie zawierał systemu Places. Decyzja ta została uzasadniona trudnościami w implementacji oraz dużą liczbą błędów, jednak nie oznacza rezygnacji [18]. Zatem twórcom wizualizacji historii pozostaje stosowanie własnych rozwiązań.

Model bazy danych Places oraz jego możliwości w porównaniu do WebMemo będą bardzo podobne. Można więc oczekiwać, że gdy wreszcie Places stanie się stabilny, wykorzystujące go rozszerzenia będą konkurować z WebMemo.

Rozdział 6

Podsumowanie

6.1. Praktyczne doświadczenia

Mozilla nie jest platformą przyjazną dla programisty. Nie posiada zintegrowanego środowiska programistycznego, które automatyzowałoby czynności niezbędne do kompilacji i uruchamiania aplikacji w platformie. Jest to uciążliwe szczególnie na początku, kiedy trzeba dokładnie prześledzić i zrozumieć ten proces.

Mozilla wykorzystuje standardowy kompilator C++ w danym systemie operacyjnym (GNU GCC w Linuksie, Microsoft Visual C++ w Windows) i udostępnia biblioteki komponentów, które dołącza się do kodu wynikowego aplikacji. Jednak w przypadku WebMemo okazało się to niewystarczające. Ze względu na ścisłą integrację z platformą konieczne było dołączanie niestandardowych bibliotek i plików nagłówkowych powstających dopiero przy kompilacji Mozilli. Również niezbędna była biblioteka SQLite.

Do rejestrowania komponentów i przetwarzania ich interfejsów służą specyficzne narzędzia. Przed zainstalowaniem i uruchomieniem aplikacji należy zbudować pakiet XPI. Podczas rozwoju WebMemo powstały skrypty wykonujące te czynności, a opis kompilacji rozszerzenia został podany w dodatku A.

Główną trudnością w przygotowaniu środowiska programistycznego dla Mozilli jest duża liczba wykorzystywanych technologii i języków programowania. Dostępne są tylko pewne narzędzia, na przykład służący do śledzenia błędów w JavaScript *Venkman* [48]. W praktyce używa się dowolnego istniejącego środowiska dla C++, a także dla JavaScript[1].

Mozilla nie posiada dobrej ani usystematyzowanej dokumentacji. Chociaż dużo informacji znajduje się w Internecie, niezbędna jest umiejętność czytania kodu źródłowego. Może to być zniechęcające ze względu na jego wielkość, jednak po pewnym czasie analiza kodu Mozilli staje się bardzo pouczająca. Przy programowaniu WebMemo szczególnie ważny był fragment dotyczący globalnej historii i źródeł danych. W ten sposób m.in. ustalono znaczenia komunikatów i słownictwa RDF.

Programista rozszerzeń powinien znać kod źródłowy platformy, ale nie może go modyfikować – nawet jeśli kilka niewielkich zmian ułatwiłoby jego zadanie. Twórcy Firefoksa są w znacznie lepszej sytuacji, ponieważ mają wpływ na kształt platformy. Przykładem jest planowany system Places.

[1] Cechą charakterystyczną aplikacji Mozilli jest niewielki kod źródłowy, który może być w całości napisany ręcznie. Przeciwieństwem jest na przykład J2EE, gdzie programowanie bez wsparcia w postaci generatora kodu byłoby uciążliwe.

6.2. Stan projektu WebMemo

Projekt WebMemo został zakończony sukcesem. Powstała aplikacja, która spełnia postawione wymagania i realizuje określone cele. Ewentualne usterki będą w przyszłości usuwane.

WebMemo w wersji 1.0 został udostępniony publicznie i znajduje się na oficjalnej stronie rozszerzeń Firefoksa pod adresem

https://addons.mozilla.org/firefox/2965.

Duże zainteresowanie użytkowników już od momentu publikacji zostało potwierdzone licznymi komentarzami na stronie projektu oraz listami skierowanymi do autora. Tym większa jest jego satysfakcja z wyniku pracy.

6.3. Perspektywy dalszego rozwoju

Można wskazać cztery kierunki dalszego rozwoju WebMemo. Pierwszy to udoskonalanie istniejących rozwiązań, tak rozszerzenia, jak i platformy. Przykładem jest wprowadzenie zaawansowanego kryterium dodawania stron WWW do historii, co w pewnych przypadkach jest złożonym problemem.

Efektywność kodu obsługi SQLite może zostać zwiększona przez grupowanie zapytań w transakcje, dzięki czemu zmniejszy się liczba zakładanych blokad pliku bazy danych. Wymaga to rezygnacji z formatu RDF, w którym trzeba niezależnie odczytywać wartości kolejnych predykatów. Wtedy z kolei pojawi się problem serializacji i zgodności z XPIDL, jeśli baza danych ma być nadal obsługiwana w oddzielnych komponentach.

Drugim kierunkiem rozwoju WebMemo jest rozszerzanie funkcjonalności, na przykład rozbudowa interfejsu użytkownika czy dodawanie własności do modelu danych. Kolejnym wymaganiem projektowym mogłaby być pełna implementacja obsługi zakładek w oparciu o historię WebMemo. Aktualnie dostępna jest tylko podstawowa funkcjonalność.

Trzeci kierunek to zwiększanie obszaru zastosowań historii WebMemo. Dwa najważniejsze to kolorowanie dowiązań[2] i dopełnianie wpisywanych przez użytkownika adresów URL. W tych przypadkach ważną zaletą WebMemo jest długi okres przechowywania danych w historii.

Jako ostatni kierunek można określić spełnianie próśb użytkowników. Na przykład w jednym z komentarzy na stronie projektu znajduje się uwaga, że WebMemo nie współpracuje z rozszerzeniem *All-In-One Sidebar* [35]. Informacje od użytkowników są bardzo cenne, ponieważ wskazują na ich rzeczywiste potrzeby i oczekiwania.

6.4. Zakończenie

Celem pracy była implementacja efektywnego systemu zarządzania i wizualizacji historii w Firefoksie. W pierwszej części została opisana Mozilla jako platforma aplikacji, na której oparta jest ta przeglądarka. Przedstawiono jej architekturę, składniki, technologie i metody programowania. Tematem drugiej było rozszerzenie WebMemo, które jest rozwiązaniem postawionego problemu. Dodatkowo dzięki znacznemu wykorzystaniu możliwości platformy jest ono dobrym przykładem programowania aplikacji Mozilli.

Realizacja projektu WebMemo okazała się być bardzo pouczająca, podobnie jak poznawanie platformy Mozilla. Pomimo bardzo wielu trudności technicznych, pominiętych w niniejszej

[2] Dowiązania do stron WWW, które były wcześniej odwiedzone i znajdują się w historii, są zwykle zaznaczane innym kolorem.

pracy, cel projektu został osiągnięty. Autor ma nadzieję, że WebMemo stanie się dla wielu użytkowników atrakcyjną alternatywą dla mechanizmu globalnej historii w Firefoksie.

Czasem uważa się, że programowania w Mozilli trudno jest się nauczyć. Jeżeli rozumie się przez to programowanie komponentów z poziomu C++, a nie tylko skryptów i dokumentów XML, to początek nauki może sprawiać wiele problemów. W pracy zostały przedstawione trudności, ale też możliwości Mozilli. W miarę poznawania platformy pierwsze tracą, a drugie zyskują na znaczeniu.

Dodatek A

Kompilacja WebMemo

A.1. Wymagania

Najlepszym sposobem przygotowania środowiska do kompilacji WebMemo jest skompilowanie Firefoksa, ponieważ w obu przypadkach wymagane jest to samo oprogramowanie. W Internecie znajduje się wiele dokumentów opisujących proces kompilacji Firefoksa [23]. Podczas jednego z etapów powstają pliki nagłówkowe oraz dynamicznie dołączane biblioteki, które są używane przez WebMemo. Alternatywnie można je skopiować z załączonej płyty CD (katalog `mozilla-sdk`).

Niezbędne jest także Gecko SDK, które zawiera podstawowe interfejsy i biblioteki (w tym XPCOM Glue). Są one konieczne dla wszystkich aplikacji, a wystarczające dla większości z nich. WebMemo wykorzystuje wewnętrzne komponenty platformy i dlatego ma większe wymagania.

Ponadto należy zainstalować bibliotekę SQLite. Jej kod źródłowy jest częścią kodu Firefoksa, a wersja binarna znajduje się na płycie CD w katalogu `sqlite-sdk`.

A.2. Kompilacja i budowa pakietu

Kod źródłowy WebMemo składa się z dwóch części. Pierwsza ma strukturę pakietu XPI i zawiera skrypty oraz dokumenty XML. Jest niezależna od systemu operacyjnego i nie wymaga kompilacji.

Druga część to napisana w sposób przenośny aplikacja C++. W systemie Linux kompiluje się ją wydając polecenie `make`. Należy pamiętać o odpowiednim ustawieniu zmiennych określających ścieżki dostępu do bibliotek i plików nagłówkowych. W katalogu źródeł znajduje się też plik projektu Visual C++ służący do kompilacji w systemie Windows. Podobnie należy ustawić wartości odpowiednich zmiennych. Biblioteki dynamiczne dostępne na płycie CD są przeznaczone dla kompilatora w wersji 7.1.

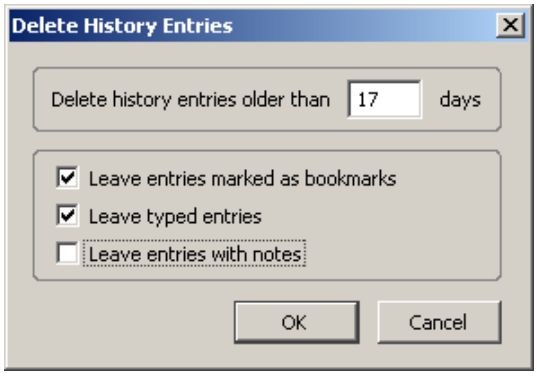
Aby kompilacja mogła się powieść konieczne jest wygenerowanie plików nagłówkowych z interfejsów XPIDL, z których również otrzymuje się plik XPT. Do realizacji obu zadań służą narzędzia zawarte w Gecko SDK.

W wyniku kompilacji powstaje `WebMemo.so` albo `WebMemo.dll`, które razem z `WebMemo.xpt` należy umieścić w podkatalogu `components` części skryptowej kodu źródłowego. Budowa pakietu polega na kompresji struktury katalogów zgodnie z formatem ZIP. Archiwum należy nadać rozszerzenie `xpi`.

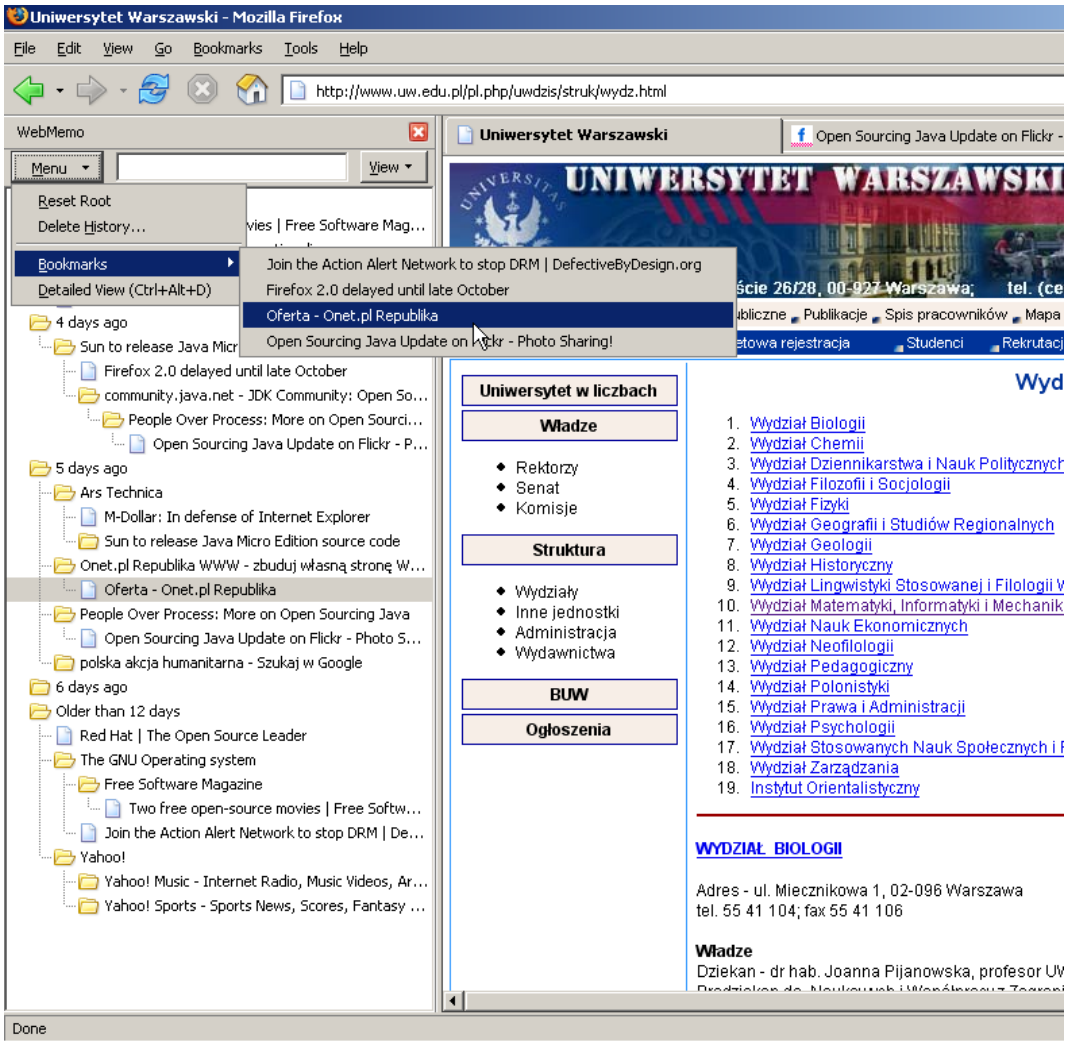
Dodatek B

Przykładowe zrzuty ekranu

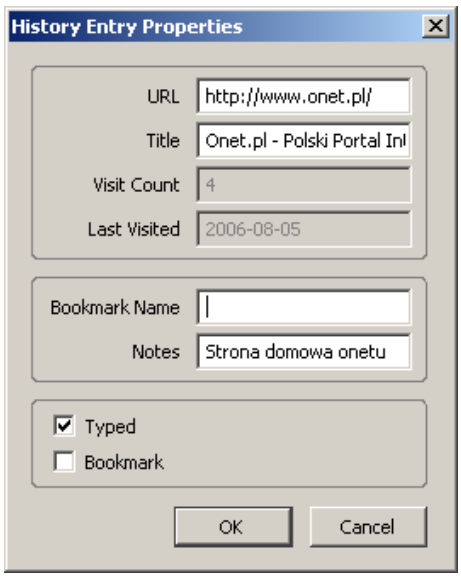
Rysunek B.1: Przykładowe drzewo historii



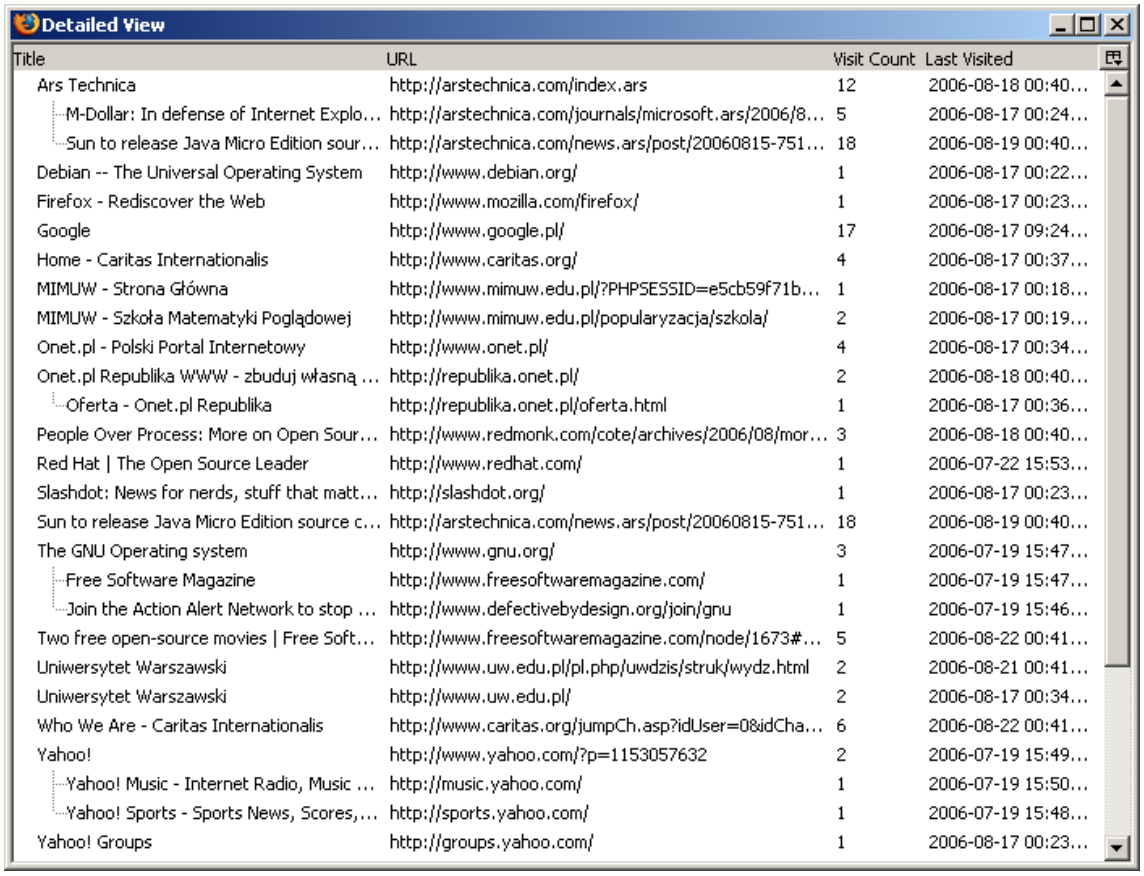
Rysunek B.2: Usuwanie historii



Rysunek B.3: Drzewo historii z otwartymi zakładkami



Rysunek B.4: Edycja właściwości elementu historii



Title	URL	Visit Count	Last Visited
Ars Technica	http://arstechnica.com/index.ars	12	2006-08-18 00:40...
M-Dollar: In defense of Internet Explo...	http://arstechnica.com/journals/microsoft.ars/2006/8...	5	2006-08-17 00:24...
Sun to release Java Micro Edition sour...	http://arstechnica.com/news.ars/post/20060815-751...	18	2006-08-19 00:40...
Debian -- The Universal Operating System	http://www.debian.org/	1	2006-08-17 00:22...
Firefox - Rediscover the Web	http://www.mozilla.com/firefox/	1	2006-08-17 00:23...
Google	http://www.google.pl/	17	2006-08-17 09:24...
Home - Caritas Internationalis	http://www.caritas.org/	4	2006-08-17 00:37...
MIMUW - Strona Główna	http://www.mimuw.edu.pl/?PHPSESSID=e5cb59f71b...	1	2006-08-17 00:18...
MIMUW - Szkoła Matematyki Poglądowej	http://www.mimuw.edu.pl/popularyzacja/szkola/	2	2006-08-17 00:19...
Onet.pl - Polski Portal Internetowy	http://www.onet.pl/	4	2006-08-17 00:34...
Onet.pl Republika WWW - zbuduj własną ...	http://republika.onet.pl/	2	2006-08-18 00:40...
Oferta - Onet.pl Republika	http://republika.onet.pl/oferta.html	1	2006-08-17 00:36...
People Over Process: More on Open Sour...	http://www.redmonk.com/cote/archives/2006/08/mor...	3	2006-08-18 00:40...
Red Hat | The Open Source Leader	http://www.redhat.com/	1	2006-07-22 15:53...
Slashdot: News for nerds, stuff that matt...	http://slashdot.org/	1	2006-08-17 00:23...
Sun to release Java Micro Edition source c...	http://arstechnica.com/news.ars/post/20060815-751...	18	2006-08-19 00:40...
The GNU Operating system	http://www.gnu.org/	3	2006-07-19 15:47...
Free Software Magazine	http://www.freesoftwaremagazine.com/	1	2006-07-19 15:47...
Join the Action Alert Network to stop ...	http://www.defectivebydesign.org/join/gnu	1	2006-07-19 15:46...
Two free open-source movies | Free Soft...	http://www.freesoftwaremagazine.com/node/1673#...	5	2006-08-22 00:41...
Uniwersytet Warszawski	http://www.uw.edu.pl/pl.php/uwdzis/struk/wydz.html	2	2006-08-21 00:41...
Uniwersytet Warszawski	http://www.uw.edu.pl/	2	2006-08-17 00:34...
Who We Are - Caritas Internationalis	http://www.caritas.org/jumpCh.asp?idUser=0&idCha...	6	2006-08-22 00:41...
Yahoo!	http://www.yahoo.com/?p=1153057632	2	2006-07-19 15:49...
Yahoo! Music - Internet Radio, Music ...	http://music.yahoo.com/	1	2006-07-19 15:50...
Yahoo! Sports - Sports News, Scores,...	http://sports.yahoo.com/	1	2006-07-19 15:48...
Yahoo! Groups	http://groups.yahoo.com/	1	2006-08-17 00:23...

Rysunek B.5: Szczegółowy widok historii

Dodatek C

Zawartość płyty CD

Na załączonej płycie CD znajdują się następujące katalogi:

- `Praca_mgr` – dokument pracy magisterskiej w formatach TEX i PDF;
- `Prezentacja` – prezentacja na temat Mozilli i WebMemo wygłoszona na seminarium magisterskim;
- `WebMemo` – projekt WebMemo, w skład którego wchodzą:
  - `Binaries` – pakiet rozszerzenia w wersji dla systemów Linux i Windows;
  - `Mozilla` – kod źródłowy i dwie wersje binarne platformy Mozilla wraz z aplikacją Firefox;
  - `SDK` – pliki wymagane do kompilacji WebMemo w systemach Linux i Windows;
  - `Scripts` – skryptowa część kodu źródłowego WebMemo;
  - `Sources` – część C++ kodu źródłowego WebMemo.

Bibliografia

Bibliografia podstawowa

[1] D. Baron, *Using XPCOM in JavaScript without leaking*, http://www.mozilla.org/scriptable/avoiding-leaks.html

[2] S. Benford, I. Taylor, D. Brailsford, B. Koleva, M. Craven, M. Fraser, G. Reynard, C. Greenhalgh, *Three Dimensional Visualization of the World Wide Web*, „ACM Computing Surveys", 31(4), 1999

[3] D. Boswell, B. King, I. Oeschger, P. Collins, E. Murphy, *Creating Applications with Mozilla*, O'Reilly Media Inc., Sebastopol, California 2002

[4] D. Carboni, *Mozilla: a development platform under the hood of your browser*, http://www.freesoftwaremagazine.com/articles/from_java_to_mozilla

[5] F. Hirsch, S. Meeks, C. Brooks, *Creating Custom Graphical Web Views Based on User Browsing History*, 6th International World Wide Web Conference, Santa Clara, California 1997

[6] W. Jones, H. Bruce, S. Dumais, *Keeping Found Things Found on the Web*, 10th International Conference on Information and Knowledge Management, Atlanta, Georgia 2001

[7] D. McCusker, *Mork Structure*, http://developer.mozilla.org/en/docs/Mork_Structure

[8] N. McFarlane, *Longhorn and Mozilla: Birds of a Feather*, http://www.devx.com/DevX/Article/17899

[9] N. McFarlane, *Rapid Application Development with Mozilla*, Prentice Hall PTR, Upper Saddle River, New Jersey 2003

[10] U. Ogbuji, *An introduction to RDF*, http://www-128.ibm.com/developerworks/library/w-rdf

[11] R. Parrish, *XPCOM: Component architecture by Mozilla*, http://www-128.ibm.com/developerworks/webservices/library/co-xpcom.html

[12] D. Turner, I. Oeschger, *Creating XPCOM Components*, http://www.mozilla.org/projects/xpcom/book/cxc

[13] D. Williams, *C++ portability guide*, http://www.mozilla.org/hacking/portable-cpp.html

Bibliografia uzupełniająca

[14] S. Aguiar, *Why XUL should stay with Mozilla*, http://aguiares.blogspot.com/2005/03/why-xul-should-stay-with-mozilla.html

[15] A. Flett, D. Baron, *Strings Guide*, http://developer.mozilla.org/en/docs/XPCOM:Strings

[16] J. Hamerly, T. Paquin, S. Walton, *Freeing the Source: The Story of Mozilla,* w: C. Dibona (red.), *Open Sources: Voices from the Open Source Revolution*, O'Reilly Media Inc., Sebastopol, California 1999

[17] M. Melez, I. Oeschger, *Mozilla Application Framework in Detail*, http://developer.mozilla.org/en/docs/Mozilla_Application_Framework_in_Detail

[18] P. Ryan, *Features cut from Firefox 2*, http://arstechnica.com/news.ars/post/20060430-6701.html

[19] S. D. Tiner, *Bird's Eye View of the Mozilla Framework*, http://developer.mozilla.org/en/docs/Bird's_Eye_View_of_the_Mozilla_Framework

[20] B. Wilson, *Browser History (Places)*, http://wiki.mozilla.org/Browser_History

[21] B. Wilson, *Browser History:Redirects*, http://wiki.mozilla.org/Browser_History:Redirects

[22] V. Vukicevic, *Mozilla2: Unified Storage*, http://wiki.mozilla.org/Mozilla2:Unified_Storage

[23] *Build Documentation*, http://developer.mozilla.org/en/docs/Build_Documentation

[24] *Hacking Mozilla*, http://www.mozilla.org/hacking

[25] *Mozilla Awards*, http://www.mozilla.com/press/awards.html

[26] *Mozilla Development Tools*, http://www.mozilla.org/tools.html

[27] *Mozilla Public License 1.1*, http://www.mozilla.org/MPL/MPL-1.1.html

[28] *SQLite: File Locking And Concurrency*, http://www.sqlite.org/lockingv3.html

[29] *Wikipedia: Application framework*, http://en.wikipedia.org/wiki/Application_framework

[30] *Wikipedia: Browser wars*, http://en.wikipedia.org/wiki/Browser_wars

[31] *Wikipedia: Mozilla Firefox*, http://pl.wikipedia.org/wiki/Mozilla_Firefox

[32] *Wikipedia: Same origin policy*, http://en.wikipedia.org/wiki/Same_origin_policy

Strony domowe aplikacji Mozilli i rozszerzeń Firefoksa

[33] *ActiveState Komodo*, http://www.activestate.com/Products/Komodo

[34] *Adblock*, http://adblock.mozdev.org

[35] *All-In-One Sidebar*, http://firefox.exxile.net

[36] *Enhanced History Manager*, http://h1.ripway.com/AnonEmoose6/FireFox/ehm.html

[37] *Epiphany*, http://www.gnome.org/projects/epiphany

[38] *FlashGot*, http://www.flashgot.net

[39] *ForwardFork*, http://forwardfork.mozdev.org

[40] *History Submenus*, https://addons.mozilla.org/firefox/682

[41] *How'd I Get Here*, http://www.squarefree.com/extensions/high

[42] *IE View*, http://ieview.mozdev.org

[43] *K-Meleon*, http://kmeleon.sourceforge.net

[44] *Mozilla Amazon Browser*, http://www.faser.net/mab

[45] *Mozilla Games*, http://games.mozdev.org

[46] *Referrer History*, https://addons.mozilla.org/firefox/1756

[47] *Session Saver*, https://addons.mozilla.org/firefox/436

[48] *Venkman*, http://www.hacksrus.com/∼ginda/venkman