PostDoc Latvia
NACIONĀLAIS ATTĪSTĪBAS PLĀNS 2020
EIROPAS SAVIENĪBA
Eiropas Reģionālās attīstības fonds
IEGULDĪJUMS TAVĀ NĀKOTNĒ

A Survey on Existing Web, Semantic Web, and Cloud Technologies

Sergejs Kozlovičs

Research #1.1.1.2/VIAA/1/16/214 "Model-Based Web Application Infrastructure with Cloud Technology Support"

Project agreement #1.1.1.2/16/I/001

Abstract

The main goal of this survey is to get insight on existing web technologies, their variations, capabilities, and shortcomings with a certain goal in mind: to adapt and re-use existing technologies within the upcoming model-based web application infrastructure (webAppOS).

The latest version of this document can be obtained at `http://webappos.org/theory`.

[☆]This document is marked as Deliverable 1.1 within Working Package 1 "Research on existing web, semantic web, and cloud technologies".

Email address: `sergejs.kozlovics@lumii.lv` (Sergejs Kozlovičs)

Contents

1. The Basics of Web Technologies

1.1. The Foundation

The history of contemporary Internet protocols can be traced back to the late 1960-s, when the Defense Advanced Research Projects Agency (DARPA) conducted the corresponding research funded by the United States Department of Defense. The work resulted in the protocol suite known as TCP/IP model. While initially it had 3 abstract layers, now it has 5 layers. The data originated at one end-point passes from the highest Application Layer down to the Physical Layer, where the data are physically transmitted. Each lower layer adds some protocol-specific overhead to the data ("packs" the data like letters are packed into envelopes). At the receiving end-point the data travels through all the layers in the reverse direction, being unpacked. While the packed data (a package) are transmitted, they can be partially unpacked and re-packed at relay stations, which we call today routers and switches. There are numerous standards explaining how routers communicate and how they know where to forward the received data.

In late 1970-ties the International Organization for Standardization (ISO) started a project, which resulted in an alternative standard of communication known as the OSI model (published in 1984). Unlike TCP/IP model, the OSI standard had 7 layers. Currently, the OSI model is more of theoretical value, where certain similarities with the *de facto* TCP/IP model can be found.

In this survey we stay in the highest layer of the TCP/IP model, namely, Application Layer (thus, we abstract from the physical transmission, which can be wired, wireless, or cellular connection; we also abstract from the connection initialization issues and transmission control, where packages can be lost and re-sent).

1.1.1. Promises

However, since certain technical information (such as IP addresses) is passed from the Application Layer to lower layers, we define some promises for webAppOS regarding the two lower layers (the Transport Layer and the Network (Internet) Layer). These promises can be considered axioms.

A promise regarding the Transport Layer:

- The TCP/IP model defines two transport protocols: TCP (which re-sends lost packages) and UDP (faster, but not guaranteed package delivery). Since webAppOS is intended to be used within the global Internet, where package losses are inevitable, we concentrate on the TCP protocol.

Promises regarding the Network Layer:

- Each connected device has one or more interfaces found on network adapters in the form of sockets, antennas, etc. Each connected interface is assigned one IP address. Each IP address can have up to 65536 sub-addresses called *ports*.

- Each IP address can be either an IPv4 address (4 bytes, usually written in the decimal form 10.0.0.1) or an IPv6 address (128 bits, usually written as 8 hexadecimal 4-digit numbers, e.g., 0000:0000:0000:0000:0000:0000:abcd:ef12; in web-browsers IPv6 addresses are enclosed in square brackets; a group of subsequent "0000:"-s can be omitted and written as "::").

- Some ranges of IP addresses are not passed through routers, these addresses are called local (usually, they are in the form of 127.0.x.x and 192.168.x.x). These IPs

can be re-used in different local-area networks without conflicts, since they are not accessible from the global Internet. The loopback local address is 127.0.0.1 (IPv4) or ::1 (IPv6).

- Global (non-local) addresses can be static (permanent) or dynamic (assigned temporarily, we cannot guarantee that the same interface will have the same IP address after the connection has been terminated or the device restarted). All global IP addresses are leased. A static address is usually assigned to a particular user on a long-term basis, while dynamic addresses are assigned from a pool of free IP addresses (often used in cellular data transmission).

- A device (we can call it a gateway) that has an interface with a global IP address (either static or dynamic) can also have additional interfaces that can be connected to other devices in a local area network (LAN) via links having a local IP address on each end. Since local IP addresses are not visible from the global Internet, there are additional steps performed at a gateway to manage such connections:

  - When a LAN device has to initialize a connection (e.g., a user opens a webpage), it connects to the gateway. The gateway associates the source IP and port of the LAN device with the global IP address and some free port on that global interface. Thus, when the gateway receives a reply, it will forward it back to the LAN device.

  - To be able to initialize connections from the global Internet to a LAN device, a port forwarding rule has to be explicitly configured on a gateway. The rule is in the form <gateway IP, gateway port, protocol TCP or UDP, target LAN IP, target LAN port>.

- While dynamic addresses and ports are capable of handling the limits of IPv4 (2^{32} values for an IP address), the Internet continues to operate on the IPv4 protocol. While some routers and switches support IPv6, others do not. To be able to use IPv6, all nodes between the source and target node have to support IPv6. It is considered that the global Internet will be eventually switched to IPv6.

- The tuple <source IP, source port, target IP, target port> defines a communication channel and is called a *socket*.

1.1.2. Sockets API

Berkeley sockets (BSD sockets, sometimes referred as POSIX sockets) is an application programming interface (API) for communication via sockets on UNIX-like systems (*NIX). Windows system use similar API called Windows Sockets API (WSA) or Winsock.[1] While sockets API can be used directly from languages like C/C++ (by including <sys/socket.h> in *NIX or <winsock2.h> in Windows plus some additional includes), other languages provide higher-level APIs over sockets. For instance, when implementing web services as Java servlets, developers work with abstract input and output streams, without thinking on low-level network issues.

[1] Windows 8 introduced Registered IO (RIO) extensions to Winsock. These extensions were intended to reduce the overhead during transition between the user and kernel OS modes.

1.1.3. Domain Names

People tend to use names instead of numbers, thus, the Domain Name System (DNS) was introduced. It associates hierarchical domains such as www.example.com (read right-to-left, delimited by commas) with IP addresses.[2] The Internet Corporation for Assigned Names and Numbers (ICANN) is the main authority that manages domain names. It authorizes others (called domain name registrars) to register domain names. Some registrars to mention are Go Daddy, Namecheap, and Fozzy; they charge several dollars a year for a second-level domain (such as yourname.org). Owners of 2nd-level domains can define subdomains on their own terms. For instance, certain 3rd-level subdomains can be obtained for free at http://www.subdomain.net/.

Multiple domain names can be associated with the same IP address. Wildcard domains (in the form of *.example.com) can also be specified. The server at the corresponding IP address can change its behavior depending on the domain name used to access its IP (if the domain name is sent with the request). Each such behavior defines a concept called a virtual host (virtual hosts can also be defined based on IP addresses and ports, not just domain names).

We are interested in domain names since webAppOS will map its web applications and services to virtual hosts associated with subdomains of the given webAppOS domain (if the webAppOS domain is specified).

1.2. Application Layer Protocols

Hypertext Transfer Protocol (HTTP) is one of the most widely used application layer protocols to deliver content to web browsers. Although the name implies textual data transfer, HTTP supports binary data as well. Development of HTTP was initiated by Tim Berners-Lee in 1989. The latest version is HTTP/2 (published as RFC 7540 in May 2015). HTTP defines several request types such as GET, HEAD, POST, DELETE, etc. Each request has protocol-specific data sent in its *headers*, while other payload can be sent in the request *body*. The "Host" header (defined since HTTP/1.1) specifies the domain name entered by the user to invoke the request. Thus, HTTP-servers are able to support virtual hosts. HTTP and its extensions are the main protocols to be used by webAppOS. Many protocols and APIs are implemented on top of HTTP.

HTTP is a stateless protocol, i.e., each query is independent on previous queries and works in the request/response manner. The state can be simulated by passing a session token either when addressing the HTTP end point, or by sending the session token in HTTP headers. A specific HTTP header relates to cookies. A cookie is a string that is associated with the particular domain and stored in a web browser. Each time a browser accesses some web page on the given domain, the cookie is appended to the HTTP request.

The URI standard (Uniform Resource Identifier) specifies the format for addressing resources by means of strings [1]. URLs (Uniform Resource Locators) form a subset of URIs that are used in Application Layer web protocols. Each URL starts with a protocol name followed by a colon, followed by a protocol-specific string that identifies the resource. In case of HTTP, the URL looks like

```
http://example.com/path?query
```

Here

- example.com is a domain name that is resolved by means of DNS;

[2] .com is the first-level domain name; example.com is the second-level domain name; www.example.com is the third-level domain name

- path is a resource location within the server; path can encode also virtual location, e.g., the resource can be generated on-the-fly;
- query is an arbitrary string for passing arguments, usually in the form name1=value1&name2=value2...

Special characters (including ":", "/", "?" as well as Unicode characters) are encoded using the URLEncode algorithm.

HTTP URL is the string the user types in the web browser in most cases (although browsers usually support other protocols as well). HTTP queries can also be executed by means of command line tools such as curl, or by connecting to HTTP end points via Berkeley sockets API.

The HTTP protocol has several extensions. Secure HTTP, HTTPS, allows the data to be encrypted via some cryptography standard (see Section 2.2). While HTTP end points usually are located on port 80, HTTPS uses the port 443 by default.

The Web Socket protocol can be used to establish and maintain a bi-directional connection between two end points for transmitting binary and/or string data with almost no overhead. URLs have the prefix "ws:". Most modern web browsers implement Web Socket API. A web socket connection is initiated as a HTTP connection, which is converted (upgraded) to a web socket. There exists also a secure version of web sockets (URLs having the prefix "wss:").

HTTP became a medium for implementing other protocols on top of it. For instance, REST is an architectural style, where HTTP path can be used to specify the function to be called (the arguments can be passed via path, query or in the HTTP request GET/POST content).

WebDAV (Web Distributed Authoring and Versioning) is a HTTP extension for accessing remote file systems. CalDAV, in its turn, is an extension to WebDAV for synchronizing calendar and timing events. CardDAV is another WebDAV extension for synchronizing contacts.

HTTP/REST can also be used for "tunelling" other protocols. For instance, versioning systems such as SVN and GIT can access their files via HTTP, while handing versioning-specific aspects by their own .

There are also other Application Layer protocols such as Telnet (port 23), SSH (Secure Shell for accessing a command line interpreter, port 22), POP (Post Office Protocol for receiving mail, port 109 or 110), etc. The File Transfer Protocol (FTP) is an interesting case of utilizing two ports, 21 and 20, for sending commands and data, respectively.

1.3. Static vs. Dynamic Web

HTML is a text markup language with the ability to link to other HTML pages. This is a real implementation of the idea brought by Vannevar Bush in 1979 [2]. While earlier the majority of HTML pages were static files at the server, today more and more pages are generated at runtime bringing dynamic web to the stage.

Dynamic server-side solutions generate HTML pages that look like static pages in the web browser. The PHP language, for instance, can be used to define HTML page templates, which are filled with the data obtained from a database such as MySQL[3]. Servlets (usually implemented in Java) are components which received HTTP requests and write HTTP responses as streams. Servlets require some server-side container to

[3]Servers utilizing Linux (operating system), Apache (web server), MySQL (database), and PHP (template language) are used so often that the term LAMP server was coined.

run (such as Tomcat or Jetty). CGI scripts is another technology - a program is called, HTTP request is passed to its standard input, and the standard output is sent back as a response.

Dynamic client-side technologies are able to change the HTML document on-the-fly. The majority of browsers support the JavaScript (ECMAScript) language[4]. A typed superset of it is TypeScript, which can be compiled to JavaScript. A recent development WebAssembly is aimed to boost performance and to allow other languages to be compiled for web browsers. However, WebAssembly currently does not support out-of-the-box garbage collector, thus, it has to be implemented in a library, which increases the load time.

JavaScript code is able to access certain functions implemented in web browsers such as functions for creating HTTP requests or for establishing web socket connections. The approach, where a JavaScript code gets data from the server via an asynchronous HTTP request is called AJAX (Asynchronous JavaScript and XML). Although initially XML was used to encode the data, currently JSON is also popular (thus the "X" in AJAX is not only XML any more). Due to security reasons, browsers usually do not allow scripts to load data via AJAX from other domains (see Section 2.1).

Google Web Toolkit (GWT) is a technology for compiling Java programs to JavaScript. With GWT, non-GUI Java programs that do not use Java libraries intensively can be compiled with minimal efforts. GUI programs and programs relying on non-primitive Java API require more serious work before they can be compiled to JavaScript. For instance, Java GUI code has to be re-written using GWT Web UI class library.

Instead of developing (and/or porting) everything in/to JavaScript/TypeScript, one can use specific frameworks such as Blazor and Angular, which factor out many technical aspects of client-server communication, and much more.

1.4. Peer-to-Peer Technologies

Certain use cases require computers to connect directly to each other, without an intermediate server. However, the topology of the Internet is designed in such a way that a request should be initiated by the client; the communication gateways recall the source IP and port and then allow the server to send the response back to the client (otherwise, the response could be blocked by the firewall). In case the client is in a private network, where network address translation (NAT) is used to map local and global IP addresses and ports, bi-directional communication is not possible right away, since local IP addresses and ports are not accessible from the global Internet (unless a request waiting for a response originated from them). However, by means of a technique called hole punching, two end points from private networks can "fool" the gateways and establish a peer-to-peer connection. This requires a special STUN[5] server during the connection initialization. However, for certain firewall configurations, STUN server is not able to "fool" the gateways. Thus, a proxy server (TURN[6] server) can be used to forward peer-to-peer requests. Although TURN server is able to ensure peer-to-peer communication in most cases, this approach requires more resources, thus, STUN servers should be used first.

An open source implementation of TURN and STUN server is available at `https://github.com/coturn/coturn`.

[4]JavaScript variables can hold data of any type (number, string, or object). All numbers, integer and real, are encoded as 8-byte IEEE doubles.

[5]Session Traversal Utilities for NAT

[6]Traversal Using Relay NAT

2. Web Security

There is plenty of information on vulnerabilities of web applications (see, for example the book by M.Andrews and J.A.Whittaker [3]). An interesting initiative led by OWASP (Open Web Application Security Project) resulted in the list "The Ten Most Critical Web Application Security Risks" [4]. In this section we mention several security issues that influence the design choices of webAppOS.

2.1. Cross-Site Scripting

An *origin* is defined as protocol+domain+port number. The *same-origin policy* is a concept that web browsers do not allow scripts loaded from one origin to call scripts (or event to load data via AJAX) from another origin. Without the same-origin policy, if the user has been logged in to some domain D, a malicious code from another domain M could perform certain operations (e.g., bank transactions) on the first domain D. This vulnerability is known as *cross-site scripting* (XSS), and is mentioned as Risk A7:2017 in the OWASP list. Although implementation details of the same-origin policy differ from browser to browser, web developers have to be aware about it: if there is a need to access data from another remote origin R, special steps have to be taken to bypass the same-origin policy. Some approaches are:

- The remote origin R can include the **Access-Control-Allow-Origin** header to specify which origins are allowed to access the data (the value "*" can be use to denote all origins).
- A HTTP proxy can be launched on the same origin D to forward requests to a remote origin R. However, R will assume that all requests are coming from D. This may lead to a ban of domain D, if there are certain per-domain limits imposed by R. In addition, a proxy introduces additional overhead.
- A universal proxy on an arbitrary domain A can be launched. The proxy would add the "Access-Control-Allow-Origin: *" header to all responses received from R. The shortcoming of the previous point also applies here.

2.2. Encryption

2.2.1. SSL/TLS and Certificates

Transport Layer Security, TLS, is a protocol for ensuring data privacy and integrity over the network. Secure Sockets Layer, SSL, is the TLS predecessor, which is now deprecated. Today TLS 1.0 is also considered insecure. Instead, TLS 1.2+ is recommended. HTTPS protocol can be referred as HTTP over TLS or SSL.

SSL and TLS use public key cryptography algorithms, which rely on certificates. The server must have the public and private key pair. The public key is sent to a client and is used to decipher server messages and to encrypt client messages. The private key is used at the server side in reverse direction. To ensure that the server certificate is authentic, it has to be signed (also via public key cryptography) by some certification authority, CA. The authority certificate can big signed by some other CA, etc., forming a chain. The web browser has to trust the root CA to validate chains of certificates automatically. The majority of web browsers come with a predefined set of trusted root CAs. Usually, issuing a certificate requires some payment, since CAs need to validate the server or the company, which will use the certificate. Certificates have to be renewed on a regular basis (e.g., once a year), perhaps, with a longer key length, since the power of computers may increase by than time.

Because of payments and efforts requires to obtain, configure and renew certificates, the (almost) 100% secure web remained just a dream for a long time. Fortunately, a recent initiative by Internet Security Research Group is a big step towards bringing dream to a reality.

2.2.2. Let's Encrypt Free Certificates and Automatic Renewal

In 2016, Internet Security Research Group initiated the Let's Encrypt service aimed to automate the process of creating and renewing SSL/TLS certificates, which could be issued and validated free of charge. The service is based on the Automatic Certificate Management Environment (ACME) protocol and is supported by many organizations including Mozilla, Akamai, Cisco, Facebook, GitHub, etc. Most modern web browsers come with Let's Encrypt root certificates, thus, no additional steps are required from the users to trust Let's Encrypt as a CA.

Since the certificates can be automatically renewed, Let's Encrypt decided to shorted the certificate renewal time to 3 months. A specific bot (like certbot, `https://certbot.eff.org/`) can be used to renew certificates, which are due to expire.

2.2.3. Configuring Certificates

Since different browsers may support different versions of SSL/TLS, the server can offer a list of available cryptographic methods known as the *cipher suite*. Since SSL up to v2 did not validate the cipher suite exchange process, intruders could force both the client and the server to use the weakest method. To avoid this weak cryptography attack, we configure webAppOS to explicitly allow only TLS1.2+ in a server cipher suite (this value will be upgraded continuously to reflect the state-of-the-art in recommended cryptographic methods). In addition, webAppOS will include a built-in certbot to renew certificates on a regular basis with a potential to increase the length of the key or even switch to some other cryptographic algorithms (e.g., to switch eventually to elliptic curve cryptography from initial RSA).

Since ACME will validate the server using the HTTP port 80, it must be accessible. In *NIX systems non-root users are not able to listen to port 80 (and other ports below 1024). Since for security reasons webAppOS should not be launched as root, we recommend to set up port 80 redirect. This can be done via a TCP-level proxy or via an Application Level proxy (in cooperation with some web server such as apache2). Table 1 summarizes these approaches (we assume webAppOS is running on host 1.2.3.4, ports 4570/HTTP and 4571/HTTPS).

In addition, one may wish to allow a non-root process ("java" in case of webAppOS) to listen to ports below 1024:

```
setcap 'cap_net_bind_service=+ep' /path/to/java
```

2.2.4. Proxying HTTPS

TCP-level proxies can just forward the received encrypted packages as is. However, if virtual hosts are used, and the proxy has to be chosen based on the particular virtual host specified with a HTTPS request, the whole HTTPS stream must be decrypted. Such proxies are called Application Level proxies. They must have access to the corresponding private SSL/TLS certificates. When certificates are being renewed, the proxy has to be informed about the new certificates. Thus, webAppOS certbot should be able to invoke a configurable command for that (e.g., to place the renewed certificates where the proxy will be able to find them). Table 2 summarizes approaches to HTTPS proxying.

Approach	Solution
TCP redirect via iptables	# /sbin/iptables −t nat −I PREROUTING −p tcp −−dport 80 −j REDIRECT −−to−port 4570
TCP redirect via ipchains	# /sbin/ipchains −I input −−proto TCP −−dport 80 −j REDIRECT 4570

Application Level redirect via apache2	First, install mod_proxy and insert apache2 modules:
	`sudo a2enmod proxy`
	`sudo a2enmod proxy_http`
	`sudo a2enmod proxy_balancer`
	`sudo a2enmod lbmethod_byrequests`
	Second, create a file within /etc/apache2/sites-enabled/some-file-name.conf with the following configuration:
	`LoadModule proxy_wstunnel_module /usr/lib/apache2/modules/mod_proxy_wstunnel.so`
	`<VirtualHost *:80>`
	`ServerAdmin admin@domain.org`
	`ServerName domain.org`
	`ServerAlias *.domain.org`
	`ProxyPreserveHost On`
	`ProxyRequests off`
	`ProxyPassMatch ^/(ws(/.*)?)$ ws://1.2.3.4:4570/$1`
	`ProxyPass "/" "http://1.2.3.4:4570/"`
	`ProxyPassReverse "/" "http://1.2.3.4:4570/"`
	`</VirtualHost>`
	Finally, restart Apache:
	`service apache2 restart`

Table 1: Redirecting port 80 in *NIX systems.

Approach	Solution
TCP redirect via iptables	# /sbin/iptables −t nat −I PREROUTING −p tcp −−dport 443 −j REDIRECT −−to−port 4571
TCP redirect via ipchains	# /sbin/ipchains −I input −−proto TCP −−dport 443 −j REDIRECT 4571

Application Level redirect via apache2	First, install mod_proxy and insert apache2 modules:
	`sudo a2enmod proxy`
	`sudo a2enmod proxy_http`
	`sudo a2enmod proxy_balancer`
	`sudo a2enmod lbmethod_byrequests`
	Second, create a file within /etc/apache2/sites-enabled/some-file-name.conf with the following configuration:
	`LoadModule proxy_wstunnel_module /usr/lib/apache2/modules/mod_proxy_wstunnel.so`
	`<VirtualHost *:443>`
	`ServerAdmin admin@domain.org`
	`ServerName domain.org`
	`ServerAlias *.domain.org`
	`SSLEngine On`
	`SSLProxyEngine On`
	`SSLProxyVerify none`
	`SSLProxyCheckPeerCN off`
	`SSLProxyCheckPeerName off`
	`SSLProxyCheckPeerExpire off`
	`# for Apache >= 2.4.8 use only SSLCertificateFile for a`
	`# combined cert+chain (SSLCertificateChainFile not needed)`
	`SSLCertificateFile /path/to/webAppOS/etc/acme/certs/fullchain.pem`
	`# for Apache < 2.4.8, use separate files`
	`# SSLCertificateFile`
	`# /path/to/webAppOS/etc/acme/certs/cert.pem`
	`# SSLCertificateChainFile`
	`# /path/to/webAppOS/etc/acme/certs/chain.pem`
	`SSLCertificateKeyFile /path/to/webAppOS/etc/acme/certs/privkey.pem`
	`ProxyPreserveHost On`
	`ProxyRequests off`
	`ProxyPassMatch ^/(ws(/.*)?)$ wss://1.2.3.4:4571/$1`
	`#variant: ProxyPassMatch ^/(ws(/.*)?)$`
	`# ws://85.254.199.113:4570/$1`
	`ProxyPass "/" "https://1.2.3.4:4571/"`
	`ProxyPassReverse "/" "https://1.2.3.4:4571/"`
	`</VirtualHost>`
	Finally, restart Apache:
	`service apache2 restart`

Table 2: Redirecting port 443 in *NIX systems.

In addition, one may wish to allow a non-root process ("java" in case of webAppOS) to listen to ports below 1024:

```
setcap 'cap_net_bind_service=+ep' /path/to/java
```

2.2.5. URL encryption

When using HTTPS, URLs are encrypted, thus, one can use URL path or query section to pass values (e.g., passwords or tokens) to the server. However, the domain part of the URL is probably not secured, since requests to DNS servers for resolving domain names are usually not encrypted. In addition, web browsers can store the whole URLs in their cache/history[7].

2.3. Authentication

2.3.1. Managing states within stateless HTTP

Since HTTP is a stateless protocol, usages requiring states have to encode the state somehow. One of the most popular approaches is to use HTTP cookies — domain-specific strings that are persisted by the web browser and automatically sent out with every HTTP request for the given domain. However, there are some issues regarding cookies:

- cookies can be used to track user's browsing history,
- websites using cookies are required by the law (e.g., EU law) to ask for user's agreement.

There is another solution, without using cookies. Each user (or browser) can be assigned a special string called access token, which is stored in the browser memory and sent as a part of request when a protected resource has to be accessed. This is the approach that will be used by *webAppOS* to manage sessions.

Another option is to use HTTP built-in authentication. However, all standard digests (that are used to avoid sending plain passwords) are too insecure, thus, this form of authentication should be used only in combination with HTTPS. This method is implemented in the web browser, thus, its user interface may be very simple. The browser stores the entered password for some time and re-sends it for requests requiring authentication. While HTTP authentication may seem outdated (i.e., it has primitive design provided by the web browser), it is still used for certain protocols such as WebDAV, where programs can provide their own GUI for entering the password. In *webAppOS*, certain services will implement HTTP authentication for compatibility.

2.3.2. OAuth 2.0

Instead of creating own authentication solution for each web server, which forces users to create yet another account, we can rely on third party authentication mechanisms (such as Google, Facebook, LinkedIn, etc.) accessible via the OAuth 2.0 protocol or alternatives. Currently, OAuth 1.0 is considered insecure, thus, OAuth 2.0 should be used. The third party server, after authenticating the user, returns a token, which can be used at the server side to check, whether the authentication was successful. The same token can be used also to access certain user-specific data (such as e-mail, calendar, and contacts), if the user allowed such access during the authentication process.

[7] See also: `https://stackoverflow.com/questions/499591/are-https-urls-encrypted`

2.3.3. Two-factor authentication

Two-factor authentication is an extension of a traditional login+password mechanism, where some additional code is required to log in. The code can be obtained via another device such as a mobile phone. On the one hand, this approach is more secure, since it is not sufficient for an attacker to know just the login and the password (they need access to the phone as well, which may be locked, etc.). On the other hand, if the phone is lost, the legal user would not be able to log in, thus, some additional recovery key is required.

Certain cloud services require two-factor authentication. For instance, when accessing iCloud Contacts, the iOS device automatically pops up a dialog with a code. Certain Windows services send an SMS with a code.

Developers can introduce two-factor authentication in their software by means of third-party services and corresponding mobile applications such as Google Authenticator or Duo Mobile.

2.3.4. Cryptographic nonce

Once an access token is obtained, it can be used to perform certain actions in the name of user. In theory, once performed, the action can be replayed again by an attacker, who intercepts all the traffic. To avoid duplicate actions (e.g., duplicate payments) that could be used in favor of an attacker, a method based on the cryptographic nonce is used. A nonce is a number that can be used to perform a request just once. In the simplest implementation, the number is increased with every request. Other implementations can use random numbers or the datetime stamp.

3. Existing Web Frameworks And Semantic Web Technologies

3.1. Frameworks for Web Applications

There are numerous web application frameworks, which are good for business-oriented form-based web applications. However, each such framework is tied to particular specific technologies (also programming languages), and a framework-specific technical configuration plays an important role in creating web applications based on the chosen framework. Many of such frameworks did not survive. Tables 3 and 4 list just a few popular frameworks.

By September 26, 2018, Wikipedia article "Comparison of web frameworks"[8] lists 62 frameworks, some of them are discontinued.

In contrast to existing framework, *webAppOS* will not impose environment-specific restrictions to web applications. The only exception is *webAppOS* itself, where drivers and other system modules will be implemented according to certain Java or JavaScript APIs[9].

In addition, when designing webAppOS, we will use the Twelve-factor methodology (`https://12factor.net/`) following the world's best practice of developing web applications.

3.2. UI Frameworks

Developing user interface (UI) for web environment differ from UI development for standalone applications. A parsed HTML page results in the Document Object Model (DOM), which can be manipulated on-the-fly by client-side JavaScript code. By means of CSS, HTML elements can be adapted to virtually any possible look and feel. There are many frameworks providing UI components for web pages as well as keyboard, mouse and touch event handling functionality.

The most prominent frameworks are Bootstrap (`http://getbootstrap.com/`), jQueryUI (`http://jqueryui.com/`), Dojo Toolkit (`https://dojotoolkit.org/`)[10], ZURB (`https://zurb.com/`), Semantic UI (`http://semantic-ui.com/`), Yahoo Pure (`http://purecss.io/`), UIkit by YOOtheme (`https://getuikit.com/v2/`), and Google Web Toolkit, GWT (`http://www.gwtproject.org/`). Google Web Toolkit uses an interesting approach of compiling Java code (describing UI components and their logic) to JavaScript, thus, GWT, in essence, is a cross-compiler that can be used to cross-compile not just UI, but virtually all algorithms implemented in Java to JavaScript.

An excellent example of commercial UI platform for the web is Sencha Ext JS[11] (`https://www.sencha.com/products/extjs/`). It is also available under the GNU GPLv3 license requiring the whole client-side application code to be licensed under GPLv3. Sencha Ext JS provides a lot of finely-tuned component with elegant design (both classic and modern).

It seems reasonable not to force web application developers to use one particular UI framework, since all have their advantages and disadvantages. Besides, a developer may

[8] `https://en.wikipedia.org/wiki/Comparison_of_web_frameworks`, `https://web.archive.org/web/20180926130641/https://en.wikipedia.org/wiki/Comparison_of_web_frameworks`

[9] These modules can rely on code written for other environments: e.g., Java can use native code by means of JNI (Java Native Interface), while JavaScript code can dynamically load Java applets, Flash animations, etc. within the web browser.

[10] Dojo 2 (`https://dojo.io/`) has been completely rewritten and uses different logic beneath its components, although Dojo 1.x components can be used with Dojo 2.

[11] In 2015 Sencha Touch merged into Ext JS.

Web application framework	**Specific features**
Node.js (`https://nodejs.org`)	Uses asynchronous calls with server-side JavaScript code. The same code pieces can be used both at the client and the server side (sometimes with minor modifications or "ifs"). Server-side asynchronous calls must be simple; long-running computation-intensive calls can block the server and create a bottleneck. Node.js modules can be used independently in static client-side applications via a translation to pure HTML+JS+CSS. This can be done by means of bundling solutions such as Rollup, Webpack, Parcel, and Browserify.
Meteor (`https://www.meteor.com/`)	Allows rapid creation of web applications in JavaScript with automatic client-server synchronization. Meteor uses a NoSQL database MongoDB, which is optimized for fast queries, but not for fast writes. Meteor requires explicit data listeners and is tied to JavaScript. Regarding scalability, Meteor is tailored to the commercial Galaxy cloud.
Google Apps Script (`https://www.google.com/script/start/`)	Allows developing web applications intended to be run in a web browser. Google Apps Script integrates with Google services (thus, for example, Google Drive can be used as a storage). However, if we need certain server-side computation or database, we need to create a web-service or some API for that.
Google Chrome OS (`https://www.google.com/chromebook/`)	Makes a browser act as an operating system, thus, making the software dependent on a particular browser.
ASP.NET Core (`https://www.asp.net/mvc`)	A cross-platform and modular successor of ASP.NET MVC and ASP.NET Web API. Uses the Model-View-Controller design pattern for creating web applications. ASP.NET-based solutions rely on a big technology "zoo" provided by Microsoft. The open-source community, while trying to ensure compatibility with non-Microsoft technologies, just adds to that "zoo". ASP.NET solutions can be easily deployed to the Azure cloud.
Apache Wicket (`http://wicket.apache.org`)	A component-based framework for stateful GUI components arranged as trees. Has integration with popular JavaScript libraries Boostrap and jQuery. It also integrates with Spring for creating automatic configurations. Apache Wicket close siblings are JavaServer Faces (`https://javaee.github.io/javaserverfaces-spec/`) and Tapestry (`http://tapestry.apache.org/`).

Table 3: Some frameworks for web applications.

Web application framework	Specific features
Blazor (`https://blazor.net/`)	Uses recent WebAssembly technology (efficient client-side runtime environment supported by major browsers). Web applications can be developed in C# and then translated to WebAssembly.
Angular.js (`https://angularjs.org/`) and Angular 2+ (`angular.io`)	A JavaScript-based framework for web applications implementing the client-side model–view–controller (MVC) or model–view–viewmodel (MVVM) design patterns. Angular.js factors out many technical aspects of client-server communication, and much more. Angular 2+ (or Angular Application Platform) is a rewrite of Angular JS utilizing TypeScript.
Django (`https://www.djangoproject.com/`)	A Python-based framework for database-driven websites. Django follows the model-view-template design pattern. The framework is very Python-centric, — Python is used even for settings.

Table 4: Some frameworks for web applications (continued).

have more skills and experience with a particular framework, thus, it could take much more time to develop and UI if another framework had to be learned. However, since certain webAppOS functionality requires some UI toolkit, e.g., for displaying the list of installed web applications or to show dialog windows, certain webAppOS modules will use the Dojo toolkit as single all-in-one solution that does not require additional libraries.

3.3. Memory for Web And Semantic Web Applications

For web applications we need some analog of traditional memory. However, that memory should be at a higher-level than just an array of bytes, since we need to track memory changes for synchronization over the network. It seems reasonable to use a graph-like structure for memory.

MOF (Meta-Object Facility) is a standard developed by OMG (Object Management Group) for describing formal models [5]. A model, in essence, is a graph of objects, links and object properties. Models usually conform to some metamodel. Metamodels, in their turn, conform to some meta-metamodel (e.g., CMOF from the MOF standard). In practice alternative MOF-like meta-metamodels (such as Java-based EMF/ECore) are used [6, 7]. A meta-metamodel usually describes itself, thus, no additional "meta"-s are needed. This is called the Three-Level Conjecture (a concept introduced by J.Bezivin et.al.) [8, 9]. Some scholars call these 3 levels linguistic meta-levels, while data levels are called ontological meta-levels [10, 11, 12]. Figure 1 shows an example, where the need for multiple ontological meta-levels occurs.

Models are usually modified by specific programs called *model transformations* written in some model transformation language (like MOLA, Lx, Epsilon, ATL, VIATRA, etc.)[13, 14, 15, 16, 17]. However, in a broader sense, any program code that is able to access MOF-like models via some API (e.g., ECore API) can be considered a model transformation.

Although existing SQL and no-SQL databases can be tamed for storing models (e.g., via object-relational mapping, ORM), such approach is more comprehensive and less efficient than using a true model repository directly [18, 19]. Examples of true model repositories are ECore and JR [6, 20]. Alternatively, graph databases such as OpenLink

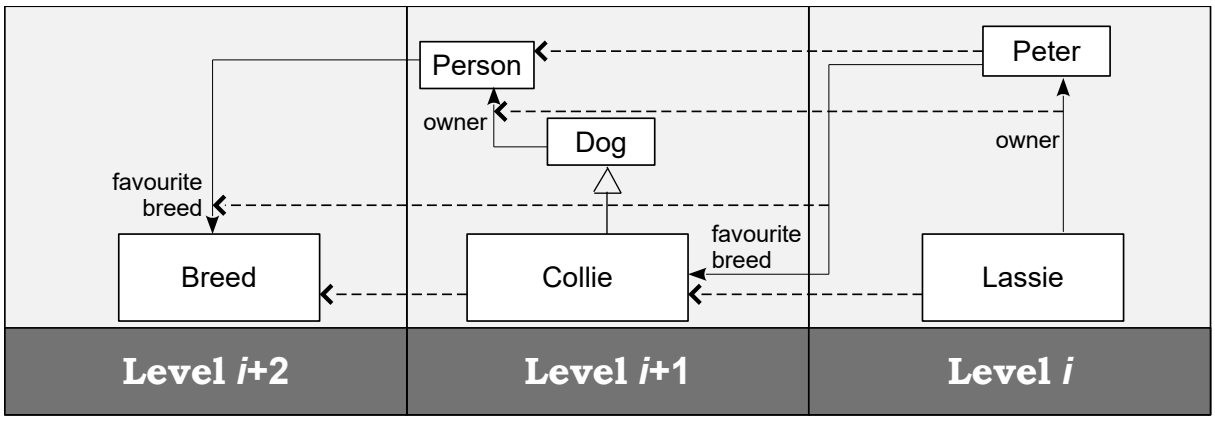

Figure 1: An example of multiple mixed meta-levels (dashed lines represent the "instance-of" relation). The relation "favourite breed" between Person and Breed as well as the link between Peter and Collie cross two adjacent meta-levels.

Virtuoso[12] and graphDB[13] can be used. They provide more capabilities than simple model repositories at the cost of performance.

Semantic Web is a set of standards for extending the existing World Wide Web with the ability to represent data in a way for automated processing. Data are represented in a graph-like structure, where formal semantics can also be described. They extend graph-like data by adding semantics. Data are encoded in triples (subject, predicate, object), which form a graph (subjects and objects are nodes, while predicates are edges). RDF is a standard containing certain predefined relations that can be used to describe triples [21]. RDF Schema is a language, which extends RDF and permits describing taxonomies of classes as RDF vocabularies [22]. In RDF/RDFS elements are identified by URIs. There are numerous interchangeable data formats for storing RDF data such as RDF/XML, Notation3, Turtle, JSON-LD, and RDFa. The latter is used to specify RDF annotations within HTML and XML documents. Semantic data can be stored on different servers, where elements of one data set can reference elements from another data set. This leads to the concept of Linked Data. An important aspect of Linked Data is the ability to execute semantic queries on interlinked data stored on different machines. The most common data sets that are referenced from other data sets are DBpedia (extracts from Wikipedia), FOAF (persons database), and GeoNames (worldwide geographical features). A community-driven project schema.org provides a collection of standardized data schemata (vocabularies) to be used with semantic data.

Web Ontology Language (OWL) bring formal semantics and additional vocabulary to RDF-like data. OWL additions ensure that certain automatic software called semantic reasoners can infer certain properties of data. The current version is OWL 2, where are 2 subsets: OWL DL and OWL Full [23]. The first subset contains the vocabulary that ensures that any OWL DL ontology is decidable. There are certain OWL DL variants (OWL EL, OWL QL, OWL RL, which can be combined) with sub-exponential complexity for certain tasks [24]. In OWL 1 there was also the OWL Lite subset that was eliminated in OWL 2 [25]. While OWL Full is undecidable, it permits having multiple meta-levels and mixing them. The following databases and repositories can be used to store OWL ontologies: Sesame [26]; Virtuoso [27]; OWLIM [28]; OWL API [29]; Apache Jena [30];

[12] `https://virtuoso.openlinksw.com/`
[13] `http://graphdb.ontotext.com/`

JR [20]; AllegroGraph [31].

Classical tasks performed by semantic reasoners over ontologies are [32]:

- Consistency checking: whether the given ontology is not contradictory.
- Concept satisfiability: whether the given class can have at least one instance.
- Classification: computes the complete class hierarchy.
- Realization: find the most specific class for the given instance.

Reasoners are based on description logic. The most prominent open-source reasoners are Pellet[14], Fact++[15], Hoolet[16], and Racer [17] (former commercial RacerPro), HermiT. There are also commercial reasoners such as OntoBroker[18] (known as KAON2 for non-commercial purposes).

Semantic reasoners infer data according to the **open world assumption** (in contrast to the **closed world assumption** used in traditional databases and logic programming). These two worlds can be describes in one sentence:

> "[The closed] world assumption implies that everything we don't know is *false*, while the open world assumption states that everything we don't know is *undefined*." [33]:

The closed world is based on the "negation as failure" principle, while the open world is based on the "negation as contradiction" principle [34]. An OWL ontology can be converted to the closed world assumption by adding some additional OWL statements.

[14] `https://github.com/stardog-union/pellet`
[15] `http://owl.man.ac.uk/factplusplus/`
[16] `http://owl.man.ac.uk/hoolet/`
[17] `http://www.racer-systems.com/`
[18] `http://www.semafora-systems.com/en/products/ontobroker/`

4. Cloud Technologies

4.1. Theoretical Background

In the document "The NIST Definition of Cloud Computing" lists 5 essential characteristics of the cloud model, namely [35]:

- on-demand self-service,
- broad network access,
- resource pooling,
- rapid elasticity,
- measured service.

There are 3 service models:

- Software as a Service (SaaS) is the highest level that brings a ready web-based application to the consumer. The application is executed within the cloud, perhaps, relying on certain application platform (see the next model).
- Platform as a Service (PaaS) is the middle level of deployment that brings certain programming languages, tools, libraries, and services for developers of web applications. On the one hand, the applications become tied to a particular platform. On the other hand, the platform factors out many implementation details, thus, web applications can be developed much faster than from scratch.
- Infrastructure as a Service (IaaS) is the lowest level that provides cloud storage and compute engines that are able to run arbitrary software (usually, as virtual machines) in the cloud.

From this hierarchy, webAppOS should be considered as open-source PaaS for developing SaaS-es. To be able to run webAppOS on different infrastructures (IaaS-es), an abstraction layer can be introduced.

Richard Stallman, an open-source pioneer, introduced a similarly-souding, but different term — SaaSS (Service as a Software Substitute) [36]. The term denotes software that could be run on a desktop computer, but for some (e.g., commercial) reasons is provided as a service running in the cloud. From Stallman's point of view, SaaSS takes away freedom since SaaSS code is usually not available for end users. Moreover, users' data have to be sent via the network, thus, the users lose control of their data, too. Being in opposition to SaaSS, Stallman suggests applying GNU Affero General Public License (AGPL) to remote software, thus, anybody who modifies such code and uses it on their own servers is forced to publish their code changes, too.

4.2. Cloud Computing

4.2.1. Cloud Infrastructures

There are numerous cloud infrastructures (IaaS-es) for launching virtual servers (virtual machines, VMs) in the cloud. The most prominent open-source infrastructure is OpenStack (`https://www.openstack.org/`), while noticeable commercial infrastructures are Google Cloud Platform (`https://cloud.google.com/`), Amazon Web Services (`https://aws.amazon.com/`), Oracle Cloud (`http://cloud.oracle.com/`), IBM Cloud (`https://www.ibm.com/cloud/`, formerly SoftLayer), Microsoft Azure (`https:`

//azure.microsoft.com), and Internap (https://www.inap.com/). An interesting feature of Oracle Cloud is that besides traditional cloud services it provides also Data as a Service (DaaS), a sibling of SaaS for data. Most cloud infrastructures are able to launch Linux and Windows virtual machines (sometimes also called instances). They are able to provide a static IP address and sometimes even a DNS name (e.g., Amazon provides a DNS name as amazonaws.com sub-subdomain).

Besides cloud infrastructures for launching fully-fledged virtual machines via a supervisor (such as XEN), there is also a lightweight container-based approach. The most prominent example is Docker (https://www.docker.com/), which uses Linux kernel feature to serve multiple user-space instances. Such approach eliminates the supervisor layer, which boosts the performance of containers. Docker integrates well with different cloud infrastructures.

4.2.2. Cloud Images

Cloud infrastructures use the same hard drive images as virtual machines (such as VirtualBox, QEMU, VMware, etc.) do. Although it should be possible to launch traditional hard disk images in the cloud, there are certain specific aspects:

- All cloud providers can usually run Linux virtual machines. Some can run Windows virtual machines, but this incurs additional license costs[19]. However, only a few cloud providers can run BSD[20] systems: the most prominent providers supporting BSD systems are open-source OpenStack and proprietary CloudSigma[21].

- Cloud images[22] designed for cloud infrastructures do not need certain drivers and boot software. Thus, each cloud infrastructure usually comes up with some predefined tailored for that infrastructure images of popular operating systems (usually, Ubuntu Linux and CentOS Linux). Some infrastructures install their own additions (similar to virtual machine add-ons) providing drivers and specific functionality such as easier exchange of SSH keys. Each time a VM is created from a predefined image, certain initialization occurs. Some infrastructures rely on the "cloud-init" standard (https://cloud-init.io/, originating from Ubuntu Linux) for initializing custom cloud instances. However, image initialization may differ in different infrastructures. Cloud-init configuration is usually passed as "user data" — a textual argument that is passed only once to the VM at the first launch time.

4.2.3. Cloud Configurations

Cloud infrastructures provide different settings for different requirements concerning virtual machines. For instance, OpenStack defines instance flavors such as m1.small (1 CPU, 2 GB RAM) and m1.medium (2 CPUs, 4 GB RAM); Amazon defines t2.micro as 1 CPU and 1 GB RAM, while t2.medium corresponds to m1.medium in OpenStack. Each VM instance usually works on a limited hard drive to store the OS and configuration (up to 20 GB). Data are usually stored in a separate disk volume, which can be easily attached to a VM instance and mounted there.

[19] Some Linuxes, like SuSE Enterprise also require a license cost.

[20] FreeBSD, NetBSD, OpenBSD

[21] https://www.cloudsigma.com

[22] The term "Just enough Operating System" (JEOS) was popular in early 2010-ties; currently the term "cloud image" is used.

4.2.4. Accessing Cloud Instances

Running VM instances are usually accessed via a secure shell, SSH[23]. The common practice is to disable password authentication via SSH, but allow connections via a private key. To configure that, just after an instance has been created, log in via a cloud provider console (e.g., in OpenStack) or a cloud provide-specific SSH client with automatically generated keys (e.g., in Google Cloud). Generate your private and public keys[24]. To store the public key, append a row to the <non-root-user-home-directory>/.ssh/authorized_keys file:

```
ssh-rsa your-public-key, e.g. AAAAB3...nM=
```

The private key must be kept secure. A password can be used to secure the private key file.

To disable SSH password authentication on the VM instance, edit the /etc/ssh/sshd_config file:

```
RSAAuthentication yes
PubkeyAuthentication yes
ChallengeResponseAuthentication no
PasswordAuthentication no
UsePAM no
```

Finally, restart the ssh service (the command may differ on different *NIX systems):

```
sudo service ssh restart
```

Long-running commands issued via an SSH session require the session to stay connected. To be able to disconnect, while still keeping the command running, the GNU screen utility can be used. When SSH session is established for the first time, launch

```
screen
```

or

```
screen -S <session-name>
```

During each next SSH session, the screen can be resumed by issuing the following command:

```
screen -r
```

To list all available screens, call

```
screen -ls
```

Alternatively, Byobu (`http://byobu.co/`) can be used.

To copy files from local PC to a VM, one of the following methods can be used:

- Launch a local HTTP server in the directory containing files to be copied to a remote server. To launch the web server, Python can be used. If Python 3 is installed, the command id

  ```
  python -m http.server
  ```

[23] SSH service usually listens to port 22, thus, it has to be open.

[24] You can use PuttyGen or openssl for that.

Cloud IaaS	Command-line tools
Amazon	Install python package awscli via python pip (for more detail see `https://aws.amazon.com/cli/`): `pip install awscli --upgrade --user` Then use the "aws" command to access Amazon services.

Google Cloud	Install Cloud Tools via apt (for Debian/Ubuntu Linux, for other systems see `https://cloud.google.com/sdk/docs`):
	`# adding Google repository and keys`
	`export CLOUD_SDK_REPO="cloud-sdk-$(lsb_release -c -s)"`
	`echo "deb http://packages.cloud.google.com/apt \`
	`  $CLOUD_SDK_REPO main" | sudo tee \`
	`  -a /etc/apt/sources.list.d/google-cloud-sdk.list`
	`curl https://packages.cloud.google.com/apt\`
	`  /doc/apt-key.gpg\`
	`  | sudo apt-key add -`
	`# updating the repository db`
	`sudo apt-get update`
	`# installing...`
	`sudo apt-get install google-cloud-sdk`
	`# initialize settings...`
	`gcloud init`
	Then use the "gcloud" command to access Google Cloud.

OpenStack	Install python-openstackclient via apt (for Debian/Ubuntu Linux): `sudo apt install python-openstackclient` Notice that python-openstackclient changed its command line arguments in version 3, thus, calls to the client v2 are not compatible with v3. After setting up certain environment variables, use the "nova" and "openstack" commands to access compute services. Use the "glance" command to access images.

Table 5: Command line clients for popular cloud infrastructures.

If Python 2 is installed, the command is

```
python -m SimpleHTTPServer
```

- Use SSH built-in file Secure Copy protocol (SCP). The "scp" command can be used in *NIX systems. On Windows systems, WinSCP can be used.

4.2.5. Cloud APIs

Usually, cloud infrastructures are accessible via HTTP REST API. Some have command-line tools, which connect to the REST API behind the scenes, but provide URL-neutral command-line interface. Table 5 lists command line clients for popular cloud infrastructures.

4.2.6. Automatic Scaling

Cloud providers start to offer automatic scaling capabilities to their clouds. For instance, with Amazon AWS Fargate (`https://aws.amazon.com/fargate/`), developers just have to provide the containers, which will automatically be managed and scaled by AWS Fargate in an optimal way.

4.2.7. Serverless Applications

Some cloud providers offer convenient services for serverless applications (applications written mostly using browser-side technologies, but requiring certain server-side functionality, such as a database). Amazon offers AWS Serverless Application Model (`https://github.com/awslabs/serverless-application-model`), while Google provides event-driven serverless platform called Google Cloud Functions (`https://cloud.google.com/functions/`).

4.3. Other Cloud Services

Finally, we mention other useful cloud services that can be re-used or re-implemented in webAppOS.

4.3.1. Static Web Pages

DNS registrars usually offer to buy certain space for static web pages (sometimes, classical PHP/SQL stack is also offered). Although, server-side technologies are unavailable or limited, the price is also low compared to launching dedicated virtual servers in the cloud.

Google Cloud also offers to host static web pages without launching a virtual machine (see `https://cloud.google.com/storage/docs/hosting-static-website`). The cost is calculated based on the storage and traffic used (usually, very low for small static web pages used by a limited group of people).

4.3.2. Cloud Drives

Cloud drives such as OneDrive, Google Drive, iCloud drive, DropBox, etc. offer a convenient way to store users' files in the cloud and to synchronize them between various devices. Free storage is usually limited to just a few gigabytes, but additional storage can be bought for a reasonable price. Different cloud drives usually implement different APIs to access the files (although some provide WebDAV interface). Commercial efforts by the CloudRail (`https://cloudrail.com`) team resulted in a well-though API for accessing various web services, including cloud drives, in a uniform way.

4.3.3. iCloud CloudKit

Apple has iCloud services that can be accessed from native iOS and macOS applications as well as web applications via CloudKit (or CloudKit JS). Apple Developer website states:

> "Use CloudKit JS to build a web interface that lets users access the same public and private databases as your CloudKit app running on iOS or macOS. You must have an existing CloudKit app and enable web services to use CloudKit JS."
>
> (from `https://developer.apple.com/documentation/cloudkitjs`)

Web Desktop	Description
Os.js (https://www.os-js.org/)	An excellent Node.js-based project.
WebDesktop.biz	Excellent commercial desktop.
AaronOS (https://aaron-os-mineandcraft12.c9.io/aosBeta.php)	Unix-like web-desktop. Currently, not downloadable to a custom web server.
System (https://system-developer-beta.000webhostapp.com/)	Windows-like web-desktop. Currently, not downloadable to a custom web server.

Table 6: Some web desktops.

Technology name	Description
Broadway (https://developer.gnome.org/gtk3/stable/gtk-broadway.html)	A technology that is a part of GNU GTK+ library. Existing GTK-based Linux/*NIX applications running on the server can stream their windows to the web browser.
xpra (https://xpra.org)	A tool for running X programs and redirecting their windows to a remote machine. Xpra can preserve the state of windows even on disconnect; it also forwards applications independently, thus, different applications do not share the common root window.
Citrix Virtual Apps (formerly XenApp and XenDesktop), https://www.citrix.com/products/citrix-virtual-apps-and-desktops/	Similar to Microsoft RemoteApp or Remote Desktop. Allows launching Windows, Linux, and SaaS applications or even whole desktops on a remote machine.

Table 7: Technologies for forwarding application windows via the network.

The limits are 25 MB of assets per user, 2.5 MB of database storage per user, 50 MB of data transfer per user, and 10 queries per second per 100K users (reasonable, if users synchronize their data just occasionally). In case the number of users exceeds 4 millions, the total limits do not increase with each new users, but are split among all of them.

4.3.4. Web Desktops and Virtual Apps

Web desktops are desktop-like environments in a web browser window. Sometimes they are called web OS-es, however, they resemble window managers, not full operating systems. As of September 2018, many are discontinued (e.g., eyeOS, ZeroPC). Table 6 lists several active projects.

There are also other technologies for forwarding application windows via the network. Some are listed in Table 7.

4.3.5. Collaboration Services

Google CollaborativeString[25] object provides a means for multiple users accessing the data (the string) to converge to a common state. Earlier, Google presented its Google

[25] https://developers.google.com/realtime/realtime-quickstart

Wave engine for collaborating on XML-like documents. Google wave was discontinued in 2012 by forwarding the development to Apache (Apache Wave project). However, the latter project retired in January 2018.

References

[1] IETF. RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax. http://www.ietf.org/rfc/rfc2396.txt.

[2] V. Bush, "As we may think," *SIGPC Note.*, vol. 1, no. 4, pp. 36–44, Apr. 1979.

[3] M. Andrews and J. A. Whittaker, *How to Break Web Software: Functional and Security Testing of Web Applications and Web Services*. Addison-Wesley Professional, 2006.

[4] OWASP, "The ten most critical web application security risks," https://www.owasp.org.

[5] Object Management Group, *OMG Meta Object Facility (MOF) Core Specification Version 2.4.1*, Object Management Group Std. formal/2011-08-07, 2011.

[6] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework, 2nd Edition*, E. Gamma, L. Nackman, and J. Wiegand, Eds. Addison-Wesley, 2008.

[7] Eclipse Modeling Framework (EMF, Eclipse Modeling subproject). http://www.eclipse.org/emf.

[8] I. Kurtev, J. Bézivin, and M. Aksit, "Technological spaces: An initial appraisal," in *CoopIS, DOA 2002 Federated Conferences, Industrial track*, 2002.

[9] J. Bézivin and I. Kurtev, "Model-based technology integration with the technical space concept," in *Proceedings of the Metainformatics Symposium*, 2005.

[10] C. Atkinson and T. Kühne, "Model-Driven Development: A metamodeling foundation," *IEEE Software*, vol. 20, no. 5, pp. 36–41, Sep. 2003.

[11] T. Kühne, "Matters of (meta-) modeling," *Software and Systems Modeling*, vol. 5, pp. 369–385, 2006.

[12] ——, "Clarifying matters of (meta-) modeling: an author's reply," *Software and Systems Modeling*, vol. 5, pp. 395–401, 2006.

[13] A. Kalnins, J. Barzdins, and E. Celms, "Model transformation language MOLA," in *Model Driven Architecture*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, vol. 3599, pp. 62–76.

[14] J. Barzdins, A. Kalnins, E. Rencis, and S. Rikacovs, "Model transformation languages and their implementation by bootstrapping method," in *Pillars of computer science*, A. Avron, N. Dershowitz, and A. Rabinovich, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 130–145.

[15] D. Kolovos, L. Rose, and R. Paige, "The Epsilon Book," http://www.eclipse.org/epsilon/doc/book/.

[16] F. Jouault and I. Kurtev, "Transforming models with ATL," in *Proceedings of the 2005 international conference on Satellite Events at the MoDELS*, ser. MoDELS'05. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 128–138.

[17] D. Varró and A. Balogh, "The model transformation language of the VIATRA2 framework," *Sci. Comput. Program.*, vol. 68, no. 3, pp. 187–207, Oct. 2007.

[18] K. Anuja, "Object relational mapping," Ph.D. dissertation, Cochin University of Science and Technology, 2007.

[19] S. Ambler. Mapping objects to relational databases: O/R mapping in detail. http://www.agiledata.org/essays/mappingObjects.html.

[20] M. Opmanis and K. Čerāns, "Multilevel data repository for ontological and metamodeling," in *Databases and Information Systems VI - Selected Papers from the Ninth International Baltic Conference, DB&IS 2010*, 2011.

[21] W3C, *Resource Description Framework*, http://www.w3.org/RDF/, W3C (a suite of recommendations) 2004-02-10.

[22] ——, *RDF Vocabulary Description Language 1.0: RDF Schema*, http://www.w3.org/TR/rdf-schema/, W3C Recommendation 10 February 2004.

[23] ——, *OWL 2 Web Ontology Language Document Overview (Second Edition)*, http://www.w3.org/TR/owl2-overview/, W3C Recommendation 11 December 2012.

[24] *OWL 2 Web Ontology Language Profiles (Second Edition)*, http://www.w3.org/TR/owl2-profiles/, W3C Std. 11 December 2012.

[25] W3C, *OWL Web Ontology Language Reference*, http://www.w3.org/TR/owl-ref/, W3C Recommendation 10 February 2004.

[26] "Sesame home page," http://www.openrdf.org/.

[27] "Virtuoso open-source edition," http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/.

[28] OWLIM semantic repository. http://www.ontotext.com/owlim/.

[29] The OWL API. http://owlapi.sourceforge.net/.

[30] Apache Jena. http://jena.apache.org/.

[31] Allegrograph RDFStore Web 3.0's database. http://www.franz.com/agraph/allegrograph/.

[32] F. Corno and L. Farinetti, "Logic and reasoning in the semantic web (part II – OWL)," Materials for the "1LHVIU - Semantic Web: Technologies, Tools, Applications" course at *Politecnico di Torino, Dipartimento di Automatica e Informatica*, 2012, http://elite.polito.it/files/courses/01LHV/2012/7-OWLreasoning.pdf.

[33] S. Mazzocchi, "Closed world vs. open world: the first semantic web battle," A blog at Stefano's Linotype, June 16, 2005, http://www.betaversion.org/stefano/linotype/news/91/.

[34] M. Zakharyaschev, "Reasoning with OWL. Open vs closed worlds. Constructors." Materials for the "Semantic Web" course, http://www.dcs.bbk.ac.uk/michael/sw/slides/Sew11-6.pdf.

[35] P. M. Mell and T. Grance, "The NIST definition of cloud computing," Gaithersburg, MD, United States, Special publication 800-145, 2011.

[36] R. Stallman, "Who does that server really serve?" https://www.gnu.org/philosophy/who-does-that-server-really-serve.en.html.