LESSONS LEARNED FROM 8 YEARS OF FIELD EXPERIENCE WITH OPEN SOURCE PLATFORM FOR COMMERCIAL WEB APPLICATIONS

Ing. Jan Vlčinský

Ing. Jan Vlčinský – CAD programy, Slunečnicová 3/338, 734 01 Karviná, Česká Republika
jan.vlcinsky@cad-programs.com

Abstrakt: Poučení z 8 let provozních zkušeností s vývojem komerčních web aplikací na platformě open source. Příspěvek shrnuje 8 let vývoje komerčních web aplikací na základě open source komponent. Všech 9 projektů souvisí s dopravními informacemi a poslední 4 pracují s mapami a geografickou lokalizací.

V první části jsou načrtnuty obchodní a technické požadavky a preference. Kde je to vhodné, jsou uvedeny výhody použití open source platformy.

Druhá část stručně popisuje 9 konkrétních projektů. Pro každý projekt je uveden účel, programovací jazyky, použité technologie a období užívání. Současně jsou formulována poučení z daného projektu s důrazem na open source technologie a někdy jsou uvedeny architektonické poznámky.

Třetí část uvádí hlavní open source komponenty, které byly v uvedených projektech použity, komentuje jejich silné a slabé stránky a vyhodnocuje vhodnost pro použití v budoucích projektech.

V poslední části jsou shrnuta obecná ponaučení.

Práce by měla pomoci každému, kdo uvažuje o vážném nasazení open source technologií pro vývoj komerčních web aplikací, na která jsou kladena přísná obchodní a technická omezení. Práce tak může být východiskem pro jasnou definici (pravděpodobně open source) vývojářské platformy.

Přesto, že poslední uvedené projekty byly postaveny na základě Javy, EJB3, aplikačního serveru JBoss a databáze PostgreSQL s rozšířením PostGIS, závěrečná poučení mohou být uplatněna na jakoukoliv jinou specifickou technologii a sadu komponent.

Klíčová ponaučení jsou: Zvolte kompletní, avšak minimální nutný počet technologií, Uvažte méně časté upgrady a Nehrňte se do nových technologií příliš brzy.

Jak vidno, uvedená doporučení nejsou úzce spjata s open source, spadají spíše do kategorie obecně platných doporučených postupů pro vývoj software.

Klíčová slova: komerční, web aplikace, open source, poučení, mapa, licence, Java, JavaScript, PHP, JMS, EJB3, AJAX, JBoss AS, JBossESB, PostgreSQL, PostGIS, Firebird, Lucene, JGraphT, JDO, fulltext

Abstract. Lessons learned from 8 years of field experience with open source platform for commercial web applications. The paper summarizes 8 years of development of commercial web applications based on open source components. All of 9 projects are traffic information related and last 4 projects works with maps and geographical localization.

First, typical business and technical requirements and preferences are outlined and where appropriate, open source advantages are mentioned.

Second part describes briefly the 9 projects. Each project has defined purpose, programming languages, used technologies and period of use. Then, lessons learned in that project are formulated with accent to open source technologies and sometime with architectural notes.

Third part lists main open source components, which were in those projects practically used, strong and weak points are commented and advice for future use evaluated.

Finally, general Lessons learned are summarized.

The work should help anyone, who is planning for serious use of open source technologies for developing web application under strict business and technical constraints. The work can server as starting point for clear definition of (possibly open source) development platform.

Though, last projects were built on Java, JBoss with EJB3, PostgreSQL and PostGIS, most of the final Lessons learned can be applied to any other specific technology and set of components.

The key recommendations are Choose complete, but minimal number of technologies; Consider less upgrades and Do not adopt new technologies too early.

As can be seen, the recommendations are not strictly bound to open source, they rather fall into category of general best practices for software development.

Keywords: commercial, Web application, open source, lessons learned, map, license, Java, JavaScript, PHP, JMS, EJB3, AJAX, JBoss AS, JBossESB, PostgreSQL, PostGIS, Firebird, Lucene, JGraphT, JDO, full text

1 Introduction

Last 8 years I and my small team spent developing traffic information related commercial web applications in my own company. Because we really enjoyed advantages of open source tools, libraries and other goodies, which I would call open source platform, I try to summarize what we learned in that period. I hope this might help others to profit from the new innovative approach, which is slowly becoming common one.
Business requirements are summarized in first chapter to emphasize constrains for efficiency and overall costs, which must be met and respected with commercial applications.
Technical requirements in next chapter define technical scope of applications described here.
Description of projects we developed using open source platform mentioned in following chapter contain not only their specification, but also formulate specific lessons we learned.
Lessons learned finally generalize all the specific lessons into set of recommendations which I believe are common to many commercial projects build on open source platform..

	period	PHP (+Firebird)			Java (+Tomcat+JBoss)		+GIS (+Jboss+RichWeb)			
	started	2001	2003	2003	2003	2005	2005	2006	2007	2007
	Application	DIS	DIZAS	TN1	NTIC	Prokop (RDS-TMC)	eRDIS	TN2	CDI2	CDI3
Application Characteristics	# of users	3	3	40	720	0	5	20	200	200
	# in systems (import)	2	2	1	0	1	0	2	2	5
	#out systems (distribute)	5	10	1	0	1	2	0	2	2
	process	YES	YES	no	YES	no	YES	no	YES	no
	distribute	YES	YES	no	no	YES	YES	no	YES	YES
	publish	no	no	YES	YES	no	no	YES	no	YES
	realtime	no	no	no	no	YES	no	no	no	no
	GIS	no	no	no	no	no	**YES**	**YES**	**YES**	**YES**
	rich web	no	no	no	no	no	**YES**	**YES**	**YES**	**YES**
progr. Lang.	**Java**				**YES**	**YES**	**YES**	**YES**	**YES**	**YES**
	PHP	YES	YES	YES						
	JavaScript	yes	yes	yes			YES	YES	YES	YES
OS	MS Windows	YES	(yes)	(yes)	(yes)	???	(yes)	(yes)	YES	YES
	Linux	YES	YES	YES	YES	YES	YES	YES	(yes)	(yes)
database	Firebird 1.0.x	YES	YES							
	Firebird 1.5.x	YES	YES	YES	YES					
	Firebird 2.x	-	-	-	-	-	-	-	-	-
	PostgreSQL+PostGIS					**YES**	**YES**	**YES**	**YES**	**YES**
AS	none (cron, scheduling)	YES	YES	YES						
	Tomcat				YES	(yes)	(yes)	(yes)	(yes)	(yes)
	Turbine				YES					
	Jboss AS					**YES**	**YES**	**YES**		
	Jboss EJB3							**YES**	**YES**	**YES**
w.s.	Web Server	IIS	Apache	Apache	Apache	Apache	Apache	Apache	Apache	Apache

Pic. 1. Overview of periods, technologies and projects

My acknowledgements shall go not only to those, involved in providing open source solutions available, but namely to my former colleagues: *Marek Vlčinský*, who convinced me trying open source platform; *Miroslav Španěl* for being excellent partner at architectural disputes, *Vladimír Hrdlík* for numerous ideas and links to existing solutions and *Aleš Daněk*, for widening our GIS expertise.

2 Business requirements and preferences for commercial application

This chapter describes possible requirements and preference for commercial web application. The purpose is not to give complete list (which is always specific to each application) but rather describe the most typical ones. Also note that real application can differ or even have completely contradictory requirements.
Read this as depicting of possible scenario and use anything, what applies to your current situation.

2.1 Licensing - Open versus Closed source model

Simply stated, use open source, write closed source and behave legal.
Prefer open source solutions. Many open source projects attract as sort of freeware and thus with no initial costs. Other advantages can be accessible information about existing bugs and problems, quick availability of solution for known problems, chance to fix things yourself, no burden of managing licenses.
Each project can be specific, read License agreement carefully to play fair and to avoid surprises.
Allow to develop closed source application. Despite of using open source platform, many managers and some clients feel better, if the application, they get developed can legally keep the source code closed and secret.
You should use components under "Commercial friendly" license, which allows that.
Strictly follow license agreements. This is not only fair, it is also a must. Breaking licenses of used components can put the whole solution into serious risk.
Just be honest and read the License agreement as first step of evaluation process.

2.2 Programmer and overall Productivity

Ignoring productivity is ignoring commercial aspect of your business.
Keep set of technologies minimal and simple. Too many languages and components are expensive as they require experts, who are difficult to replace, and consume more time to learn.
This task is difficult and requires good estimation of your projects and clients for some years in advance. Choosing components of the platform also require very careful evaluation. Be careful and learn from others or ask experts with background in development of commercial applications.
Reliable components. Using components, which are not reliable spoils motivation of programmers, destroy your image at clients and create extra costs.
The advice is the same as in previous point.

Provide enough functional tools. Integrated IDE, viewers for data, load testing suites, modeling software and other tooling can influence productivity a lot.
Do not be afraid to use tools, which are paid, if it returns back in productivity. We used many open source tools, but took advantage of Enterprise Architect to support modeling of the system and were all the time running desktops with MS Windows.

Platform must not require too much fine tuning. Programmers are often enthusiastic about open source projects, but if they spend too much time playing and fine tuning, they become expensive.

Platform stable in time. Every change (upgrade, replacement of a component, adoption of a new technology) is time consuming.
Evaluate likehood of component availability and support in future. Strong and alive community or commercial company behind the project makes the chance bigger.

Programmers for given technology are available. Any member of a team might be replaced in future (for whatever reason). If you are unable to find another programmer with required skills, then you have big problem (deciding for new technology, rewrite, dropping the product...).
Do not rely on one expert, choose common technologies, for which it s likely to find programmers with enough skills and knowledge.
Prefer simple and stupid solutions, if they work.
Be warned by word "nice" (technology, product, feature).

Reachable technical support.. Every software has bugs and require some maintenance. Doing it yourself is time consuming.
Community support often works quite well, but it can change and vary a lot.
Prefer components with support options available.
If you are under time pressure and have trouble getting enough programmers, paid technical support might be reasonable solution.

Stick to standards to allow component exchange. Component exchange might happen for any reason – user preferences, discontinued development, missing scalability features etc.
Design clear interfaces and use standard protocols, technologies and formats. This approach makes possible exchange easier.
If it is feasible, consider using closed source components.

3 Technical requirements and preferences

This chapter continues depicting requirements and preferences, this time technical ones.
The paper is concentrated on web application of middle size with some GIS features. For examples see last four projects we have developed (eRDIS, TN2, CDI2 and CDI3).

3.1 General Requirements

These requirements come from the nature of Web application itself.

Web application. The application will be used via web browser.
Using client-server architecture of application running in web browser simplifies tasks like installation and updates.
Pure web browser (no add-ins). Application must not require any additional plug-ins or add-ons, it may rely only on pure JavaScript web browser.
Maintenance of any extras going beyond default installation of web browser is complicating situation for all users, administrators and application provider.
Nowadays, with AJAX being matured technology on most web browsers this requirement can be fulfilled safely. For more see section 5.8 AJAX*.*
Data persistence. Data must be persisted on server.
Prefer design patterns, which allow exchanging database server. Technologies like JDO or JPA are well supported also by open source solutions like JPOX (for JDO and JPA), Hibernate and JBossEJB3 (JPA). For more see section 5.4 Object-relational mapping (ORM)*.*

3.2 Environment

The application will be placed into environment of your client and shall fit there seamlessly.

Integrate with client Identity system. Client might have some Identity system in place (typically Active Directory or LDAP) and prefers integration with it. Application shall allow for such integration.
Use JAAS technology, which is well supported in JBoss and integrates with Active Directory or LDAP.
If requirements go beyond currently supported features, like OpenID or special authorization patterns, prefer writing your own JAAS module over coding some special solution.
Generally, design your application with separated Identity system in mind even if you have to deliver the Identity system yourself.
Run in isolated Intranet. This requirement might prevent using existing Internet solutions like Google Maps etc. This was the case for CDI2 and CDI3.
Interact with outer systems. Application shall export information to or import information from other systems.
Design the interaction as independent module of your application.
Be careful with Web Services and make sure you are not running into compatibility issues.
Consider using JBossESB – it offers very modular architecture, which is easy to modify.
Make sure, that protocol and formats used are defined, complete and are really working and stable in time.
Provide logging facility to ease resolving data exchange problems.
Consider providing GUI showing exchange records as it eliminates need for administrator to dig the information from file system on server.
Run on MS Windows as well as on Linux. Client has often internal policies for operating systems in use and might require or prefer to use MS Windows server.
Selecting parts of your platform prefer those, which run on Linux as well as on MS Windows.
Most (but not all) open source projects are meeting this requirement.
Prefer languages like PHP, Java over Dot Net (which is not yet mature on Linux).
We used databases like Firebird and PostgreSQL, later one being more robust.

3.3Map related

Show interactive maps. Map layers shall be static; some content (e.g. traffic information and POI) shall be displayed dynamically and must allow user interaction.
We wrote our own web map client. For more see section 5.10 Web map client
Show custom maps. Client has often his own maps, like Global Network (aka StreetNet), DMU 25, Location tables for RDS TMC etc. and wants them to be used.
Dynamically generating maps via some WMS server is not recommended due to load predictability and stable application performance. For more see section 5.9 OGC WMS – Web Map Service.
Full text search in maps. Finding a position in a map is mostly the key factor for user productivity. Full text search shall allow finding places in map easily and quickly.
Apache Lucene is perfect solution in Java; just plan the indexes to contain geographical position to have complete result ready without additional search in database. For more, see section 5.12 Full text.
Allow updating maps few times a year. Expect that map data can be updated and this change shall be reflected in the application without big problems.
We decided for statically generated map tiles and had to develop procedure for updating the tiles according to new map data.
Do not forget to include generation of all other map data dependent things like full text indexes etc.
Even if map data claims to follow GDF 4 standard, we always found structural changes in data model even for new version of product with the same name.
Each map data contain errors, be ready to check common problems and be ready for small corrections.
If you are not data agency, do not commit yourself to repairing all mistakes in data, this is endless work.
Deegree has better support for OGC standards and allows better control over outlook of resulting maps.
Generating usable maps using WMS servers is possible, but be prepared to live with some degree of label collision issues.
If you are generating first version of maps, be aware, that to design usable outlook of maps takes a lot of work and iterations. Consider outsourcing this task.

3.4User friendly Interface

Provide Rich web UI. Nowadays, rich web interface becomes de facto standard. The more complex forms, the more important to avoid too many page reloads.
For more see section 5.8 AJAX.
Notify user about server events. As data change on the server, each relevant user shall be notified and page content shall be properly updated.
For lower level of traffic use simple polling, each few seconds as needed.
For bigger number of users and faster response times there might be problem with enough sockets and connections. Some sort of Asynchronous Request processing might help, see GlassFish ARP, Jetty ARP, Tomcat 6 NIO and AHS by IceFaces [16].

3.5Performance

Handle 50 concurrent users on today HW. Expect up to concurrent 50 users Application shall be enough responsible and if possible, shall not require too complex architecture (replicated servers etc.).
If you are not forced, prefer simpler architecture without clustering, replications etc. Simpler is better.
Optimize by using dedicated servers (web server, database server, application server).
Optimize by controlling number of concurrently processed requests. One thread is too few. Unlimited number of threads is dangerous as it can destroy your resources. Consider accepting requests, storing requests in JMS and processing them only in limited number of threads (testing needed, but often 3 to 5 threads performs best).
For profiling PostgreSQL use pgFouine log analyzer. To finetune JBoss application, try Jopr and do not overlook Hibernate Statistics.
Failover capability. Application must not rely on in-memory data and shall gracefully survive unexpected restart of server without losing data, ignoring or double processing started events etc.
Use JMS with message persistence; take advantage of transactional support in database or on application server level.
JBoss AS supports transactions natively.
Using database without transactional support is not recommended.

Archive and clean old data. Data, which are not of importance, shall be automatically exported to an archive and removed from main database. This is a must for live systems, which are expected to work for years.
Consider clean up procedures even if not externally required.
Archiving data, consider using some standard format (e.g. Atom which allows for storing custom objects, some blog systems are good examples of this approach).
Optimize for limited bandwidth. Your client might have restrictions on available bandwidth, which might be lower, then "everyone is used to". With showing maps, what often require download of large number of images, this constrain might be very hard.
Use cashing, where reasonable.
Provide request response compression, Apache web server and Tomcat can do that very well transparently with good control over specific behavior of each browser.
Use some sort of HTTP debugging Proxy (e.g. freeware Fiddler) to learn, what is really consuming the bandwidth.
Scalability. If there is good chance that use of the application might grow above of small set of role dedicated servers, demand for general scalability might appear.
JBoss AS is designed for clustering in mind and it is told to be easy.
Most of open source databases support some sort of scalability and failover features (PostgreSQL, MySQL…).

3.6Manageability

Easy license management. The number of users, servers or installations might change. Make sure, that this change does not break existing (typically paid) licenses or that there is simple mechanism, how to recognize needed licenses and get them.
If you build from freeware, there is simply no paid license to manage.
Allow monitoring parameters. If client has his own monitoring system, connec the application to that system, as it makes application administration easy and effective.
Application shall expose some of parameters for monitoring.
Use some standard interface (JMX, HTTP) to make the information available.
Be aware of security issues if you expose JMX, because JMX might allow many actions (incl. remote server shutdown).
Design usefull monitoring parameters in advance, code them and implement as it simplifies production use as well integration testing..
Provide monitoring system. If client does not have its own monitoring system, provide one.
We had the best results with Nagios. Unfortunately, it is available only on Linux and UNIX platform.

4 Our commercial applications based on open source platform

Picture 1 in Introduction has outlined projects, which were developed using open source platform. All of the projects were developed by my team and I was personally responsible for delivery of all of them.
This chapter describes those projects in more detail.
Projects are sorted chronologically.

4.1 DIZAS

Specification: System allows importing, processing and distribution of traffic information by operators (3) at call center. Information is localized by text with reference to enumerated district, no maps used.
Written in PHP 4, using database Firebird 1.0.x
Runs on MS Windows with IIS web server, later ported to Linux and Apache web server.
Some complex forms were written as Java applets.
Imports and exports are processed using scheduler from MS Windows.
Used since 2001 till 2004.
Lessons learned: PHP allows building complex solutions, but has limits, we met later on (missing background processing, bigger projects are difficult to maintain, e.g. Java is forcing more tidy style). PHP upgrades can bring surprises in changes of API, e.g. way of XSLT transformation came through bigger changes. However, on hardware we had that time PHP was the only solution.
Firebird (originally Interbase) works well to certain scale, with growing number of transactions and records, performance drops. However, on hardware we had that time we had no other choice (MySQL was missing referential integrity) and performance was good enough for many years.
Java Applets are tricky. They require JRE to be installed on client computers and this makes often compatibility issues as client systems upgrade in years. Generally, any plug-in or applet on web browser is a risk. However, browsers were not yet supporting JavaScript command XMLHttpRequest, so AJAX solution was not yet available.
Apache web server works better then IIS: IIS and PHP went broken with almost any security update of the system. Changing to Apache web server later on was very positive and reliable solution even on MS Windows.

4.2 DIS

Specification: DIS is rewrite of DIZAS.
Information localized by text with possible use of stationing on highways.
System can produce some map images based on prepared image templates.
Database upgraded to Firebird 1.5.x
Runs on Linux using Apache web server.
Scheduling using cron, later on using custom Java applications.
Used since 2003 till 2005, some parts distributing traffic information are in use up to date.
Lessons learned: **Linux on server** is stable. Migrating from MS Windows to Linux stopped never ending server restarts – services were simply running, or could be restarted only individually without need to restart the whole system.

4.3 TN1 – Traffic News v1

Specification: System for publishing traffic information to registered users, who after log in can view and filter latest traffic information incl. recorded telephone calls reporting them.
Written in PHP 4, using shared database with DIS (Firebird 1.5.x).
Extensive use of in-house component IDC, which allows SQL and XSLT templating.
Used since 2003 up to date.
Lessons learned: **PHP and Firebird** works nice for this type of “preview simple data” application. The system is simple, not using much resources. The only problems were caused by sharing the database with DIS as DIS was sometime overwhelming the system.

4.4NTIC – National Traffic Information Center

Specification: System for entering and sharing of traffic information by over 200 subjects with over 700 registered users.
Written in Java, using Apache Turbine Web Application Framework run inside of Tomcat.
Data stored in Firebird 1.5.x using JDO technology (Triactive JDO).
Run on Linux.
Used since 2003 till 2005.
Lessons learned: Java as a language is forcing to code in clean and tidy way (comparing to PHP), what makes it well suited for larger and continually expanded systems.
Apache Turbine framework was surprisingly very functional solution for developing web application of that scale. Nowadays I would recommend some more popular framework (because of more likely support in future).
Triactive JDO for solving object persistence using JDO technology was excellent solution, which simplified development greatly.

4.5Prokop (RDS-TMC)

Specification: Real time system for broadcasting Traffic Information for navigation systems using RDS TMC technology. Receives Traffic Information from outer system and manage encoding and sending data to encoder for broadcasting. System is able to mix the RDS TMC stream with another RDS stream.
Written in Java, run inside JBoss AS.
The server part is run on MS Windows or on Linux. The RDS module runs on Linux, MS Windows platform was not yet tested at production.
Data stored in PostgreSQL using JDO (Triactive JDO).
Used since 2005 up to date in two pilot tests and two full time services.
Lessons learned: **Java and JBoss AS** is well suited for real time with messaging. Original feeling, that "such a big thing" must consume a lot of resources appeared to be wrong. In fact, the real time module, which process many events each second under rather strict time constraints is run on the cheapest 512 MB model of HP Proliant server which was at 2005 available.
Java and real time works excellently: our tests proved that the application meets exactly the processing speed, which is by design of outer encoder available.
PostgreSQL on MS Windows works. Because of client requirement we had to run the server module on MS Windows. This was the first time we could use PostgreSQL, which was that time ported to MS Windows in native mode. There were not troubles with that.
JDO pays back in database portability: Thanks to using JDO technology, we could start development on Firebird database and meantime switch without any problems to PostgreSQL.
LTS Linux distribution would be better: on one installation we decided for Fedora distribution. Later on, we run out of period, when that version was supported. Choosing some LTS, even paid distribution would be wiser and would save some effort. Systems tend to be running longer, then one originally expects.

4.6eRDIS

Specification: Complete rewrite of DIS with real localization in maps.
Exports Alert-C data for RDS TMC broadcasting.
For legacy distribution sends data to DIS.
Written in Java, runs inside JBoss AS.
Data stored in PostgreSQL with PostGIS extension.
Originally used Deegree as WMS, but for performance reasons rewritten to use pre-generated static map tiles. Map client was developed in-house and is using pure JavaScript enabled web browser.
Full text search in maps using Apache Lucene.
Path finding based on JGraphT.
Used since 2005 up to date.
Lessons learned: **DWR and AJAX** works well. With DWR we can easily call a Java object methods from JavaScript. Within years DWR turned to be stable and allowing to do almost anything.
JBoss AS proved suitability for web applications. We stopped coding most of background tasks and could concentrate more on "real" work with business logic, front end etc.
JMS, ActiveMQ and JBossMQ: After some testing and carefulness with JMS technology we started to

trust it. Internal design of application became much simpler and background processing got more reliable and predictable. Originally we were experimenting with standalone JMS implementation Apache ActiveMQ. There were some minor problems, which were solved by turning completely to JBoss AS with JBossMQ included.
Deegree WMS was used to generate maps for GUI. Original design of real time map generation had to be changed to use pre-generated map tiles. This change allowed running with less resource and being more stable and responsive. Generating all map tiles (for all scales) for whole area of Czech Republic takes couple of days and generates more than 4 GB of images. Using MNU Map Server gives similar results.
Lucene for full text search: Speed of Lucene full text search is astonishing even on large index files. At the same time it does not consume much resources.
JGraphT for path finding: Searching for path finding Java solution we got good references about JGraphT. The task is finding a path in a graph with more than one million edges. With some optimization (pre-generating the graph, loading it into memory and controlled multithreading) we could offer responsiveness within fraction of a second.
PostgreSQL and PostGIS: As spatial database we used PostgreSQL with PostGIS. Support for topological tasks, queries and usability in some of GIS tools is very good.

4.7TN2 – Traffic News v2

Specification: Next generation of Traffic News for publishing traffic information to registered users with option to show them on maps.
First EJB3 application.
Written in Java, run inside JBoss AS with EJB3 container.
Data stored in PostgreSQL with PostGIS extension using JPA based on Hibernate implementation.
Full text search in maps using Apache Lucene.
Path finding based on JGraphT.
Used since 2006 up to date.
Lessons learned: **Adopting EJB3 too early**: We decided to go EJB3, which was offered by JBoss that time. We had better listen to warnings, that it is not fully EJB3 certified [21]. Solution was not yet mature that time and we spent about three months fixing numerous small problems related mostly to JPA implementation. We succeeded, but it was costly.
Apache XMLBeans: To standardize use of XML data we decided for using XMLBeans. JAXB turned not to be mature enough that time (on JBoss with that time officially unsupported Java SDK 1.6). XMLBeans have simplified many tasks since that time.

4.8CDI2 – Centrum Dopravních Informací etapa II

Specification: For detailed description see [56]
System allows creation and distribution of traffic information by operators (200) at Czech Police. Information is localized in maps and events are described using Alert-C protocol. Data are exported to JSDI (central store of traffic information in Czech Republic) and to CDI3 (see below).
Written in Java, runs inside JBoss AS with EJB3 container.
Data stored in PostgreSQL with PostGIS extension.
Map tiles are pre-generated using UMN Map Server. Map client was developed in-house and is using pure JavaScript enabled web browser.
Shows maps based on StreetNet (GDF 4.0 compatible) and DMU 25 (army digital model of Czech Republic).
Full text search in maps using Apache Lucene.
Path finding based on JGraphT.
Used since 2008
Lessons learned: JPA already works. This time was EJB3 platform ready: JBoss debugged the implementation and we already knew, how to use it Note, that the fully EJB3 certified JBoss implementation became available in December 2008 (in JBoss AS 5) [21][22]. The real EJB3 usability in JBoss came more than 20 months sooner.

4.9CDI3 – Centrum Dopravních Informací etapa III

Specification: For detailed description see [56]
System allows utilization of all available traffic information inside of Czech Police. There are expected 200 daily users.
Imports data from CDI2 (see above) and JSDI, allows for interactive filtering and preview of information in maps and allows automatic notification of information as they appear.
Based on the same technology as CDI2.
For complex order processing uses JBossESB.
For on demand task scheduling uses Quartz.
Used since 2008
Lessons learned: JBossESB: We decided to separate complex order processing from core application and used JBossESB for that. The same could be achieved by JMS, but with ESB we got much more prepared architecture and solution, which allowed us to build the solution from many small, independent parts and modules.
Smooks: Smooks is framework for transforming streams of data in various formats and is used also inside of JBossESB. However, the version incorporated that time into JBossESB was one version older and did not offer enough functionality for us. Our effort to upgrade Smooks inside JBossESB turned to be too complicated and we had to forget about using Smooks there. Nowadays Smooks support in JBossESB seems to be much better.
Quartz is scheduler and we used its sophisticated functionality to perform scheduling of sending traffic information reports, which were to be delivered in time, chosen by a user. The architecture and functionality of Quartz is excellent.

5 Featured open source projects

This chapter mentions most open source components, we were using in our applications. The list is concentrated on those components, which become part of final delivery to the client.
The list does not deal with other areas like IDE for coding, test suites, bug tracking, project management software, and office suites etc.
In some cases there are projects, which we were not using, but which appear as likely replacement or alternative. In that case this fact is explicitly stated.
Prefered solutions are mentioned first.

5.1 Operating system

Linux is well known open source operating system. We were using following distributions: Red Hat 7.2 and 7.3, Fedora Core 3, Debian 3.0 Woody, Ubuntu 5.10 Breezy Badger. Ubuntu 6.06 LTS Dapper Drake. All distributions perform well.

When deciding for distribution, consider following aspects.

1) *Distribution you are used to* (Debian and Ubuntu have a bit different directory structure and habits then Red Hat and Fedora)
2) *Outsourcing base system administration* (hardware, kernel, core services, database, monitoring) might be very effective solution (we used Active24.cz dedicated server). Accept the distribution prefered by your provider.
3) *Drivers for exceptional hardware* you might be going to use (not recommended), check availability of drivers for that hardware.
4) *Prefer LTS* (Long Time Support) versions as one can expect application run for years and unsupported version could become a problem.

5.2 Programming Language

Java is programming language and software platform, developed by Sun Microsystems and is designed to run on almost any hardware.

Comparing to PHP Java is suitable for bigger projects, because it forces writing modular code.

We used Java SDK 1.4, 1.5 and 1.6 from Sun.

Generally there were no serious problems, upgrades between versions went smooth.

The only problem was with blocking queue, when under special scenario (no data being sent in, only checking queue for available items) we run out of memory. The bug was repaired in next SDK version.

PHP was the language we started with. For smaller and simpler applications works well and does consume little resources. Even middle size project was written in PHP; however, code was more difficult to manage.

Later on we decided to drop PHP and concentrated on Java to simplify our development environment.

Upgrades to any new versions were a little bit tricky, mainly because implementation of XSLT transformation was changing.

5.3 Database systems

PostgreSQL is robust and if vacuum clean procedure is done properly, performance is very good even on growing number of records.
As PostgreSQL was made available in native MS Windows mode, it became our preferred database.
If experiencing performance problems, consider checking, if indexes are not missing (that was problem in Hibernate, not in PostgreSQL), check for vacuum procedure and use pgFouine to analyze log files.

PostGIS is extension, allowing effective use of spacial data. Because PostgreSQL supports so called GIST indexes, spacial queries are very effective. PostgreSQL with PostGIS became our preferred solution for any GIS and map related applications.
Firebird was our first database. That time (2000) it was natural choice, because it run on Windows as well as on Linux and was fully supporting referential integrity (MySQL was not).
It has very simple installation and database maintenance, unless you run into bigger volume of transactions and records.
Sometime we experienced problems with Firebird overwhelming system (Linux).
We used versions Interbase 6, Firebird 1.0.x and Firebird 1.5.x. It is likely, that Firebird version 2.x performs much better, but that time we already decided for PostgreSQL.
Because Firebird does not support R-Tree (GIST like) indexes, it is not suitable for spacial analysis (this applies to Firebird version 2.x too).

5.4Object-relational mapping (ORM)

Managing data persistence of objects using JDBC is time consuming and shall be avoided whenever possible. Loosing few percents in theoretical performance is worth comparing to lost programmer productivity (boring work, bugs, being database dependent, being in fact less effective from application performance point of view) in scale of hundreds of percents. That is why we always used some technologies like JDO or JPA.

Java Persistence API (JPA) in JBoss EJB3 with Hibernate is standard ORM implementation available in JBoss EJB3. We even succeeded to map custom spatial data types.
Hibernate allows ORM without byte code modification (as opposite to JDO) and makes build procedures simpler.
However, JDO is still offering some features, which are not possible or easy in Hibernate.
As JPA allows for using different persistence implementations, consider using some JDO implementation (JPOX) as option to Hibernate.
However, as we decided to keep JBoss platform, Hibernate has an advantage of being natively there out of the box.
Triactive JDO (JDO) was our first ORM choice and in fact it could work quite well up to date. The only disadvantage was extra step in build procedure (byte code enhancement).
Changing from Java SDK 1.4 to 1.5, enhancing tool support changed and build procedure went broken. Final solution that time required compilation of classes using older compiler or using extra enhancer from JPOX project.
Triactive JDO project became inactive for quite a time. There are some signs of ongoing activity last days (November 2008), but I would recommend JPOX today as being in time much more mature and feature complete.
JPOX (JDO) was never really used in our projects, however, for JDO I would recommend it as preferred solution nowadays.

5.5Application Server

Here I use term Application server in wider meaning as an environment offering set of services and API helping easier development andf deployment of an application.

JBoss AS is our application server of choice. We hesitated to use it because we had the feeling, that something so complex must consume a lot of resources. The true is that consuming 80 or 120 MB of RAM is not too much today. In fact, one installation of Prokop (RDS TMC), which is quite “busy” with threads, real time processing and some web interface is *running on server with 512 MB RAM in total (incl. PostgreSQL server)*.
Tomcat was our first choice and we use it up to date as it is part of JBoss AS.
Apache Turbine is a framework we used for NTIC project with quite a number of users. The NTIC was run using Turbine inside of Tomcat.
Turbine supports MVC design pattern very well and was excellent solution for us that time.
Nowadays I would think of frameworks like Spring, Struts, pure JSF or JBoss Seam (as we are JBoss like).

5.6Messaging (JMS)

To start using JMS is not easy. It looks complex and it is clear, that if it is part of an application, then it will be one of keystones, which must not break. Popular solution is „we have better write something ours“ but I strongly discourage trying that. JMS implementations are available, very well functional, tested and finally simple to use. Spend time learning JMS instead of writing some half made half featured solution.

As you start with JMS, you get many advantages like simplified modularized architecture, failover capability (with persistent messages), and controlled number of threads for processing tasks etc.

JBoss Messaging is current implementation of JMS in JBoss designed for performance, clustering and scalability. I recommend using it.
JBossMQ is preceding JMS implementation of JMS at JBoss. Was working well but nowadays prefer new JBoss Messaging. Good point is, changing implementation does not affect code almost at all.
ActiveMQ is JMS implementation by Apache and was first one we were evaluating. We had small problems and (happily) ended with JBoss AS. If you think of running without JBoss, ActiveMQ might be one of matured options.

5.7Messaging (ESB)

As applications live some years, they grow in number and complexity and aspect of data exchange between external and internal systems becomes more visible. Enterprise Service Bus approach might simplify this task as it draws exchange complexity out of single applications and creates independent solution, which can bear all the changes and reconfigurations isolated from rest of systems.

ESB might be seen as a sort of JMS extension as it handles exchange of messages.

JBossESB (1) is relatively new product of JBoss. We have evaluated some other options (e.g. Apache Camel and Mule) and decided for JBossESB 4.2.1 because it was based on JBoss, we were used to.
Architecture is very clear and once we got into it, we were building solutions like playing with Lego bricks. As it was rather young version, we had to develop some our components for reading data from read only FTP. Smooks, used in JBossESB as transformation engine, seemed very promising. It seemed that we can populate JavaBeans from XML data simply by configuring Smooks, but Smooks version used in JBossESB was one version behind and updating it in JBossESB failed and we had to use alternative solution. However, with JBossESB we were able to separate rather complex part of CDI3 application out and results were good.
Mule claims to be the most widely used open source ESB solution. We did not use it, but consider evaluating it, as it might be quite mature.

5.8AJAX

AJAX allows for developing Rich Web UI. There are three possible approaches in Java:

1. write all JavaScript code and create web interfaces in Java as you need
2. write some JavaScript and use DWR to interact with selected JavaBeans
3. Use some JSF AJAX enabled framework. This might allow for writing Rich Web UI without witting single line of JavaScript code.

Standard gadget libraries and frameworks can make the work easier and can tidy up design of your UI. At the time we evaluated them (2006), we did not find the proper one. Nowadays there are some, which seem matured enough, among others IceFaces and Exadel RichFaces.
However, be sure, you have some solution for controls you need and which are missing in the library.
Using JSF technology with AJAX framework can simplify UI development; one can create a Rich Web form without writing a line of JavaScript code.
Consider paid solution, because it can strongly influence amount of work and overall impression and functionality. JBoss Developer Studio might be the case.
DWR (Dynamic Web Remoting) is Java library, which allows for calling JavaBean methods directly from JavaScript. DWR functions very well and is widely used.
IceFaces is one of JSF AJAX enabled frameworks we tested. Because JSF seemed running slower then DWR, we used JSF forms only for rarely used forms.
RichFaces (Exadel) might be my choice today, as it is integrated into JBoss Developer Studio.

5.9OGC WMS – Web Map Service

Web Map Server is needed to provide visual representation of maps to map client.

We originally created maps dynamically, but because of performance and stability of final application we turned to pre-generated map tiles. Off line generation of map tiles works well with Deegree or UMN Map Server, but expect it takes few days, generates big number of small files (10000 and more) and takes more than 4 GB of data for area of Czech Republic (we came down to scale 1:6400).

Deegree was our first choice. Support for OGC standards is very good and we also use Deegree v 1 in another application to generate maps for WAP pages dynamically. Second version of Deegree seemed quite promising. It had wider support for new versions of specifications, but was a little bit slower. There was also quite a delay between version 1 and final release of version 2. However, nowadays the project seems again alive.
UMN Map Server was the choice of our partner to generate next version of map tiles. Performance of Deegree and UMN Map Server is very similar.

5.10 Web map client

To show maps to user, some web map client is needed. We were evaluating available open source solutions at 2005 and finally decided to write our own.

Nowadays the situation is better. New evaluation of solutions like Google Maps, OpenLayers and others would be handy.

5.11 Templating

By templating I mean some tool, which allows defining template for dynamic generation of text output. It is used to generate HTML or WML pages as well as a tool to generate text messages and e-mails.

Apache Velocity is very popular solution and we used it in some projects.
JSP can be seen as templating option too. Velocity and other templating tools claims they prevent using too much flow controls and force authors to write only tidy templates isolated from controller code. Regardless of that, I have seen Velocity templates, which were breaking this advantage and were full of control instructions making template messy. I would consider using simple JSP with agreed set of rules supported by some procedure, which would discover use of outlawed constructions.

5.12 Full text

Full text search solutions allow finding needed record or iteming quickly.

When planning for map upgrade procedure, do not forget you have to update index files too.
Lucene (Java) is excellent and very fast full text library. To use it in real situation one must find or write some tools for that. We wrote our own tooling for full text search of roads, cities and districts in Czech Republic and were very satisfied.
Lucene is only part of solution (the engine), you have to find or develop tools around (for indexing etc.). I have seen some solutions integrating Hibernate with Lucene. As objects were persisted, Lucene index was always updated. This might be handy sometime.
mnoGoSearch (C) was our option for PHP projects. PHP has some libraries for mnoGoSearch. Be aware, that mnoGoSearch is freeware only on Linux, solution for MS Windows is paid. Nowadays I would consider using some Lucene solution even from PHP to keep the set of used technologies small (remember, we were Java based last years).

5.13 Scheduler

To run some regular jobs, one needs some scheduler. But there might be more sophisticated scenarios, like sending an e-mail with dynamic report at the time, user specified at his report order.

Quartz is another excellent library. It allows any sort of scheduling and supports even dynamic task scheduling with persistence. Based on Quartz we built processing of report orders, where user could create order, set up report time, change it and delete the order.
cron is well known Linux scheduler. For platform independence I generally recommend to use your own Java based scheduler whenever possible. You may write your own or use the one, which comes with your application server.

5.14 Geo and topo calculations

JTS Topology Suite (Java) is library for geometrical calculations in geomatics and is very good. However, it lacks native support for some coordination systems and projections used in Czech Republic and we had to do this using PostGIS functions.
JGraphT (Java) is an open source Java graph theory library we used for path finding. We were able to develop solution, searching 600 000 road segments (because they are often bidirectional, resulting graph had over 1 million edges) real-time for multiple users.

5.15 Monitoring

Delivered application shall be running. With some monitoring tools one has very good chance noticing problems sooner than user and having current and historical status information at hand to resolve problems quickly.

Nagios was the system we used. The only disadvantage is, the server part of Nagios is purely Linux based and cannot run on MS Windows. However, monitoring Linux as well MS Windows works, there are many plug-ins ready and development of new ones is simple (as plug-in is command line program returning some string and result code). Architecture and concept is very simple but complete, it prevents many fake alarms, handles dependencies and can show, where source of the problem is.
Do not be mistaken that it does not show many graphs and fancy graphics, we have learned, that the status (UP, DOWN or UNREACHABLE for hosts and OK, WARNING, CRITICAL and UNKNOWN for services) is just the information, one needs to know.
There are some nice looking solutions based on Nagios (e.g. GroundWork Monitor), but they seem too complex to me (e.g. Nagios does not require database and PHP, GroundWork does).
Monju is project, solving monitoring JBoss parameters by Nagios. We did not use it, because we wrote our own Nagios plug-in for JBoss before the Monju project became available.
Be always careful with exposing MBean methods as they can be used for many actions including server shutdown.
Jopr is monitoring solution (JBoss Operating Network) open sourced recently (October 2008). Features look very promising, especially for JBoss and Java, it offers monitoring down to use of EJB3 entities, configuration of JBoss, IIS, Apache, Tomcat; restarts, Hibernate statistics, high availability and it runs also on MS Windows. I would recommend Jopr for development monitoring, as it can show many profiling type information, and would use it for production monitoring on servers with enough memory and power. However, for general monitoring I would prefer Nagios as it allows more with less power.
Anyway, take it as my opinion, as I do not have used Jopr yet.

6 Lessons Learned

This chapter summarizes general lessons learned while developing our commercial applications.

6.1 Open source platform works

Open source does not mean second hand. Many real projects have proved that open source platform allows for professional solutions.

6.2 Choose commercial friendly license

Some licenses like GNU GPL require source code distribution with binaries. Of course there are ways, how to keep the application code rather private with these licenses (distribute source code only to your clients, use the solution only in-house), but with commercial friendly licensing you do not have to solve this topic at all.
Examples of commercial friendly licenses are: GNU LGPL, Apache 2.0 License, and BSD License. GNU GPL might be used if it declares so called Linking exception [58].
In situations, the system can be delivered independently and not as part of big solution, GNU GPL might be acceptable (e.g. Linux, Nagios etc.)

6.3 Open source is not always freeware

Despite of the fact, many open source projects are available for free, this is not a rule. There are many open source projects, which provide you with a source code, however, you have to pay for use of it.
This situation is more likely with products based on some GNU GPL component.
Example: Nagios is great monitoring tool, available for free. A project Ground Work Open source offers enhanced solution based on Nagios. Two of three editions are not for free.

6.4 There are many open source components available

World of open source components is really rich and one can find here many top grade solutions, which are used even in commercial platforms.
When selecting a component, be careful and evaluate carefully regardless of open or closed source origin.

6.5 Use complete, but minimal number of technologies

Too many programming languages, technologies and software components is expensive.
Forget about continually selecting "the most appropriate" approach, rather select "the standard, preselected" solution, if usable.
Too heterogeneous platform makes maintenance difficult, complicates documentation, increases dependency on experts and prevents easy replacements of programmers in the team.
Example: We started with PHP scripting language. Later on, we thinking of using other fancy languages like Python and in some situations Perl.
Later on, as we had to switch to Java, we were happy we kept only PHP, because even those two languages (Java + PHP) were troubling enough (e.g. trainee from India knew some Java, but not PHP and she were not able to maintain the old code).

6.6 Adopting too early is expensive

The fact that a platform offers some support to a new technology does not mean it is production ready. Open source solutions are more likely to provide such a limited support and they usually do declare this fact and warn you in advance.
If you are too eager and adopt the solution too early, you may spend months fighting numerous bugs and troubles. Together with learning new technology your project might slip quite behind your original schedule.
Example: In project TN2 we decided to use EJB3 provided by JBoss. There were many small problems in new type of persistence, which cost us almost three months endless debugging. If we were wait 3 or 5 months, we would get JBoss EJB3 debugged and really ready for EJB3 almost without any effort.

6.7 Consider less upgrades

Upgrading to the latest version of a component require some testing. With application servers the changes and tests might be rather extensive. Just be aware, it costs time and money and do it only, if you have to.
With some products (e.g. JBoss), you can purchase support contract, which will provide long term support for specific version (you get only patches for known problems). The support seems often quite expensive, but compares it with costs of time of your programmers. If the paid support saves you enough time, then it could be considered as hiring a programmer from software provider to your team.
Anyway, it might complicate licensing and I do not have real experience with this type of solution.
Example: One of our products was installed on Fedora distribution of Linux. The product was used for years and the version of Fedora was not supported any more. Choosing for some (even paid) LTS distribution would save us some effort and troubles.

6.8 Expect your solutions to grow

If you do your business well, you will experience growth of your applications further, then you would originally expect.
Number of lines o codes, classes, tables, records in database, load on your application will increase and you will soon appreciate enterprise features like scalability, transactions.
Today cost of supporting these feature is not so high (CPU and RAM are rather cheap) comparing the cost of rebuilding your platform in future.
Example: Originally, we started with database Firebird. Later on number of transactions and records increased and we found PostgreSQL better performing. Maintaining two databases added to complexity of our environment.
However, until PostgreSQL became natively available on MS Windows, we did not have much choice without ignoring portability to MS Windows.
Example: Originally we started Java applications on Tomcat. Later we found we need JMS server for Messaging and added ActiveMQ. Even later we turned completely into JBoss AS with built in transactions, JMS, persistence, JMX and many other enterprise features. If we were not too afraid of wasting memory and CPU power (it was rather mind limit then real one), we could jump directly to JBoss AS and could save some platform iterations.
With JBoss AS platform is even ready for scalability.

6.9 Evaluation process is crucial

There are always some internal and external forces trying to change your platform. If you develop standard criteria and procedure, you will be able to either say quickly qualified NO or know what steps to go to make your final decission. Consider aspects like long term platform and application vision, prefered licence type, quality of support, alternative solutions, level of support etc.

6.10 Time is Money

With open source in mind, one can easily forget that license cost is not all, what has to be paid and spends too much time "playing" with any possible system.
Develop a set of rules, which helps you to stop. For example if type of license does not fit, do not evaluate.
Realize, that if you can achieve the same with paid component in less time and with equal total costs, the paid "expensive" one would be probably more advantageous from your commercial point of view, because you could earn some time.

6.11 Learn Bug tracking your Platform

One of big advantages is, that bugs are discovered quickly and solution discussed openly.
Many closed source platforms try to hide their problems, what makes any evaluation of components and technologies before real use risky and often impossible.
Learn using JIRA and other bug tracking tools of your selected component and check your version of component regularly. Do the same with some (not all) blogs.
But be careful and do not spend on it too much time.

6.12 Try to be OS Independent

If feasible, try to be portable.
Client can require (for good reasons, like keeping OS, which is well known to his IT department) another platform, then your preferred.
If your solution is not portable, you are very likely to lose your client.

6.13 Try to be Database Independent

The same as for OS apply for database. Many clients have a rule, that the database must be MS SQL or Oracle.
Using technologies like JDO, Hibernate or JPA together with portability in mind might make the change easier.
However, be prepared to spend some time on porting and testing.

6.14 Consider ready components over home made

Some programmers prefer their own home made components over standard ones, because they feel better control over it and are afraid, that if they spend time learning some "too bulky and too complex solution", they will finally learn, it does not work.
Situation should be evaluated carefully, because first running version of homemade component is not mostly the final one, completing and debugging takes time, hidden bugs are more likely and there is mostly no documentation and tutorials for it.
With standard component, one can get ready solution, which was tested by many users and have already available documentation incl. how-tos, tutorials etc. The last simplifies adopting the solution by other members of the team too.
Example: We needed some sort of simple cache to optimize performance. Cache was developed in-house and works well. However, the same could be probably achieved with JBoss built in cache, which we had to learn later on anyway.

6.15 No paid license management = simple license management

If you use freeware licensing (at least for the components, which go to the client), think twice, before adopting components requiring paid licensing, otherwise you lose one of advantages (no paid license management = simple license management).
However, if advantages of purchasing paid license out weight disadvantages, just be pragmatic.

7 Conclusions

Eight years of filed experience with open source platform proves, that open source is a platform, which for sure allows for building top notch web applications with GIS.
However, without being careful and keeping in mind the business point of view, projects can fail regardless of the chosen platform.
From all the Lessons learned I would emphasize

1. Choose complete, but minimal number of technologies
2. Consider less upgrades
3. Do not adopt new technologies too early

In one sentence: Clear vision and careful evaluation of changes are crucial.
As can be seen, the recommendations are not strictly bound to open source, they rather fall into category of general best practices for software development.

8 Terms and abbreviations

Table 1. Terms and abbreviations.

Term	Category	Description
ActiveMQ [5]	software	JMS server
Apache HTTP Server [2]	software	web server
Apache Tomcat [3]	software	servlet container (web server)
Debian [45]	software	Linux OS
Deegree [20]	software	web map server
DWR [57]	software	librabry for AJAX javascript – JavaBean calls
EJB3 [18]	technology	Enterprise JavaBean: architecture for enterprise Java applications.
Enterprise Architect [47]	software	Desktop tool for UML
Fiddler [24]	software	HTTP debugging proxy
Firebird [13]	software	database server
GDF 4 [29]	specification	GIS format for automotive navigation systems.
StreetNet [9]	map data set	Map data set for Czech Republic (GDF 4).
Hibernate [33]	software	ORM library for persistency of Java objects
IceFaces [15]	software	JSF implementation
JAAS [51]	technology	framework for pluggable Authentication and Authorization in Java
Java [48]	progr. lang.	cross platform object oriented programming language
JAXB [14]	specification	Java architecture for XML Binding
JBoss Developer Studio [39]	software	IDE for Jboss programming based on Eclipse
JbossAS [36]	software	Java Application Server
JbossEJB3 [37]	software	EJB3 container implementation
JbossESB [41]	software	Enterprise Service Bus implementation
JbossMessaging [40]	software	JMS server
JbossMQ [34]	software	JMS server
JgraphT [28]	software	graph theory library for Java
JMS [17]	technology	Java Message Service API for sending messages
Jopr [35]	software	Monitoring and Management platform for JBoss
JPA [18]	technology	Java Persistence API, framework for object persitency
JPOX [19]	software	Implementation of JDO
JSF [49]	technology	JavaServer Faces – web application framework
JSP [50]	technology	JavaServer Pages – technology to generate HTML pages
JSR220 [18]	specification	JSR for JPA
JTS Topology Suite [55]	software	Java library for topological calculations
Lucene (Java) [4]	software	Java library for fulltext indexing and searches
mnoGoSearch [23]	software	software for fulltext indexing and searches written In C
Monju [53]	software	simple framework for monitoring JBoss by Nagius
Mule [25]	software	Enterprise Service Bus implementation
MySQL [26]	software	database server
Nagios [27]	software	software for monitoring network and computer systems
OGC Standards [29]	specification	Set of Open Geospacial Consorcium standards
OpenID [30]	specification	standard for decentralized user identification
pgFouine [44]	software	utility to analyze PostgreSQL log files

Term	Category	Description
PHP [54]	progr. lang.	cross platform scripting language
PostGIS [42]	software	PostgreSQL extension supporting geographic objects
PostgreSQL [32]	software	database server
Quartz [31]	software	Java scheduling library
Red Hat Linux [38]	software	Linux OS
RichFaces (Exadel) [11]	software	JSF implementation
Smooks [12]	software	library for transforming data (XML and others)
TJDO (TriactiveJDO) [46]	software	Implementation of JDO
Turbine [6]	software	Web application framework
UMN Map Server [43]	software	web map server
Ubuntu [8]	software	Linux OS
Velocity [1]	software	Java templating library
XMLBeans [7]	software	framework for binding XML and Java objects

9 Reference

1. Apache Software Foundation. http://velocity.apache.org/engine/releases/velocity-1.5/, What is Velocity?
2. Apache Software Foundation. http://httpd.apache.org/, Welcome! - The Apache HTTP Server Project
3. Apache Software Foundation. http://tomcat.apache.org/, Apache Tomcat
4. Apache Software Foundation. http://lucene.apache.org/java/docs/features.html, Apache Lucene - Features
5. Apache Software Foundation. http://activemq.apache.org/, Apache ActiveMQ -- Index
6. Apache Software Foundation. http://turbine.apache.org/, Apache Turbine Web Application Framework
7. Apache Software Foundation.. http://xmlbeans.apache.org/, Welcome to XMLBeans
8. Canonical Ltd. http://www.ubuntu.com/, Ubuntu Home Page | Ubuntu
9. Central European Data Agency. http://www.ceda.cz/index.php?option=com_content&task=view&id=15&Itemid=41, StreetNet+TMC
10. ERTICO. http://www.ertico.com/en/links/links/gdf_-_geographic_data_files.htm, GDF - Geographic Data Files
11. Exadel, Inc. http://exadel.com/web/portal/openSourceProducts, Exadel - Open Source Products Developed with JBoss/Red Hat
12. Fennelly, Tom at. al. http://www.smooks.org/, What is Smooks
13. Firebird Project. http://www.firebirdsql.org, Firebird - The RDBMS that's going where you're going
14. Glassfish community at java.net.. https://jaxb.dev.java.net/, jaxb: JAXB Reference Implementation
15. ICEsoft Technologies Inc. http://www.icefaces.org/main/home/index.jsp, Home - ICEfaces.org
16. jack.van.ooststroom. http://www.icefaces.org/JForum/posts/list/7800.page, AHS vs Tomcat 6 NIO
17. Java Comunity Process community. http://www.jcp.org/en/jsr/detail?id=914, JSR 914: Java Message Service (JMS) API
18. Java Comunity Process community. http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html, JSR-000220 Enterprise JavaBeans 3.0 - Final Release
19. Java Persistent Objects (JPOX). http://www.jpox.org/, Java Persistent Objects (JPOX)
20. lat/lon and the GIS Research Group of the Department of Geography of University of Bonn. http://www.deegree.org/, deegree - Free Software for Spatial Data Infrastructures
21. Labourey, Sascha. http://sacha.labourey.com/2008/09/15/jboss-as-is-now-ee5-certified/, JBoss AS is now EE5 certified!
22. Labourey, Sascha. http://sacha.labourey.com/2008/12/05/as-500-we-are-done-next/, AS 5.0.0: we are done. Next.
23. Lavtech.Com Corp. http://www.mnogosearch.org/products.html, mnoGoSearch Products
24. Microsoft Corporation. http://www.fiddlertool.com/fiddler/, Fiddler Web Debugger - A free web debugging tool
25. MuleSource Inc. http://mule.mulesource.org/display/MULE/Home, Home - Mule - Open Source ESB - SOA and Integration Platform
26. MySQL AB, 2008 Sun Microsystems, Inc.. http://dev.mysql.com/downloads/mysql/5.0.html, MySQL 5.0 Downloads
27. Nagios Enterprises, LLC. http://www.nagios.org/, Nagios: The Leader and Industry Standard in Enterprise System, Network, and Application Monitoring
28. Naveh, Barak at. al.. http://jgrapht.sourceforge.net/, Welcome to JGraphT - a free Java Graph Library
29. Open Geospatial Consortium, Inc. http://www.opengeospatial.org/standards, OpenGIS Standards and Specifications
30. OpenID Foundation. http://openid.net/what/, OpenID: What is OpenID?
31. OpenSymphony. http://www.opensymphony.com/quartz/, Quartz - Quartz Overview
32. PostgreSQL Global Development Group. http://www.postgresql.org/, PostgreSQL: The world's most advanced open source database
33. Red Hat Middleware, LLC. http://www.hibernate.org/, hibernate.org – Hibernate

34. Red Hat, Inc. http://www.jboss.org/community/docs/DOC-10525, Community: JbossMQ
35. Red Hat, Inc. http://www.jboss.org/jopr/, Jopr About the project
36. Red Hat, Inc. http://www.jboss.org/jbossas/, JBoss Application Server
37. Red Hat, Inc. http://www.jboss.org/jbossejb3/, JBoss EJB3
38. Red Hat, Inc. http://www.redhat.com/rhel/server/, redhat.com | Enterprise Linux-Open Source Application for Servers built on Linux
39. Red Hat, Inc. http://www.jboss.com/products/devstudio, JBoss Developer Studio
40. Red Hat, Inc. http://www.jboss.org/jbossmessaging/, JBoss Messaging - Enterprise Open Source Messaging from Red Hat
41. Red Hat, Inc. http://www.jboss.org/jbossesb/, JBossESB - Reliable SOA infrastructure
42. Refractions Research. http://postgis.refractions.net/, PostGIS : Home
43. Regents of the University of Minnesota. http://mapserver.gis.umn.edu/, Welcome to MapServer
44. Smet, Guillaume. http://pgfouine.projects.postgresql.org/, pgFouine - a PostgreSQL log analyzer
45. Software in the Public Interest, Inc. http://www.debian.org/, Debian -- Univerzální operační systém
46. SourceForge, Inc. http://tjdo.sourceforge.net/, TJDO Project Page
47. Sparx Systems Pty Ltd. http://www.sparxsystems.com.au/products/ea/index.html, Enterprise Architect - UML Design Tools and UML CASE tools for software development
48. Sun Microsystems, Inc. http://java.sun.com/, Developer Resources for Java Technology
49. Sun Microsystems, Inc. http://java.sun.com/javaee/javaserverfaces/overview.html, JavaServer Faces Overview
50. Sun Microsystems, Inc. http://java.sun.com/products/jsp/, JavaServer Pages Technology
51. Sun Microsystems, Inc. http://java.sun.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html, JAAS Reference Guide
52. Sun Microsystems, Inc. http://java.sun.com/javaee/technologies/persistence.jsp, Java Persistence API
53. Tamale Software, Inc. http://monju.sourceforge.net/monju.html, Monju: Monitor JBoss Servers with Nagios
54. The PHP Group. http://www.php.net/, PHP: Hypertext Preprocessor
55. Vivid Solutions Inc. http://www.vividsolutions.com/jts/jtshome.htm, JTS Topology Suite
56. Vlčinský, Jan. Processing and utilization of digitally localized traffic information at Czech Police. *GIS Ostrava 2008*, Ostrava 2008. ISBN 978-80-254-1340-1
57. Walker, Joe. http://directwebremoting.org/, Direct Web Remoting
58. Wikimedia Foundation, Inc. http://en.wikipedia.org/wiki/GPL_linking_exception, GPL Linking Exception