California AHMCT Program
University of California at Davis
California Department of Transportation

LITERATURE REVIEW OF NATIONAL DEVELOPMENTS IN ATMS AND OPEN-SOURCE SOFTWARE*

Michael T. Darter, Kin S. Yen, Bahram Ravani, &

Ty A. Lasky, Principal Investigator

AHMCT Research Report
UCD-ARR-06-12-08-01

Interim Report of Contract IA 65A0210 - Task Order 06-22

December 8, 2006

Affiliations:
The authors are with the AHMCT Research Center, Department of Mechanical & Aeronautical Engineering, University of California, Davis, CA 95616-5294

* This report has been prepared in cooperation with the State of California, Business and Transportation Agency, Department of Transportation and is based on work supported by Contract Number 65A0210 - Task Order 06-22 through the Advanced Highway Maintenance and Construction Technology Research Center at the University of California at Davis.

Technical Documentation Page

1. Report No. F/CA/RI-2006/10	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Literature Review of National Developments in ATMS and Open-Source Software		5. Report Date December 8, 2006
		6. Performing Organization Code
7. Author(s): Michael T. Darter, Kin S. Yen, Bahram Ravani, and Ty A. Lasky		8. Performing Organization Report No. UCD-ARR-06-12-08-01
9. Performing Organization Name and Address AHMCT Research Center UCD Dept of Mechanical & Aeronautical Engineering Davis, California 95616-5294		10. Work Unit No. (TRAIS) 11. Contract or Grant IA 65A0210 - Task Order 06-22
12. Sponsoring Agency Name and Address California Department of Transportation Division of Research and Innovation 1127 O Street Sacramento, CA 94273-0001		13. Type of Report and Period Covered Interim Report December 2006 14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

Advanced Traffic Management Systems (ATMS) have successfully and impressively improved roadway traffic safety, reduced congestion, and increased economic productivity. As ATMS build-out continues into lower-density population areas, ATMS cost can become a serious obstacle, slowing implementation and negatively affecting safety, congestion, and productivity. This report explores the feasibility of using open-source software and commodity computer hardware to lower ATMS implementation costs. The objectives of this report are to:

- Summarize the history and current developments in ATMS software and hardware, and their relation to open-source software and commodity multi-source x86 hardware,
- Summarize the history, strengths, and weaknesses of open-source software, and
- Summarize relevant ATMS hardware and software projects that use open source, or are open-source projects themselves.

In preparing this report, the authors surveyed and analyzed relevant articles in research journals, reports, and industry news sources. In addition, individuals were contacted from state DOTs and research institutions that were involved with open-source ATMS projects. Based on the information that was collected and analyzed for this report, the authors found that a number of ATMS and ITS applications have been developed and deployed using open-source software. These systems have a broad range of features, capabilities, and significant operational use. Potential benefits include low initial and recurring costs, customization ability, security, stability, and reduced lock-in. Benefits provided by OSS and its unique development model must be balanced with a consideration of concerns such as technical knowledge of personnel, compatibility with existing software, problems with multiple operating system distributions, and documentation.

17. Key Words Advanced Traffic Management Systems, Intelligent Transportation System, Open-source Software, Open Source Software, commodity hardware, OSS, FOSS, ATMS, ITS	18. Distribution Statement No restrictions. This document is available to the public through the National Technical Information Service, Springfield, Virginia 22161.

20. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 132	22. Price

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized
(PF V2.1, 6/30/92)

ABSTRACT

Advanced Traffic Management Systems (ATMS) have successfully and impressively improved roadway traffic safety, reduced congestion, and increased economic productivity. As ATMS build-out continues into lower-density population areas, ATMS cost can become a serious obstacle, slowing implementation and negatively affecting safety, congestion, and productivity. This report explores the feasibility of using open-source software and commodity computer hardware to lower ATMS implementation costs. The objectives of this report are to:

- Summarize the history and current developments in ATMS software and hardware, and their relation to open-source software and commodity multi-source x86 hardware,
- Summarize the history, strengths, and weaknesses of open-source software, and
- Summarize relevant ATMS hardware and software projects that use open source, or are open-source projects themselves.

In preparing this report, the authors surveyed and analyzed relevant articles in research journals, reports, and industry news sources. In addition, individuals were contacted from state DOTs and research institutions that were involved with open-source ATMS projects. Based on the information that was collected and analyzed for this report, the authors found that a number of ATMS and ITS applications have been developed and deployed using open-source software. These systems have a broad range of features, capabilities, and significant operational use. Potential benefits include low initial and recurring costs, customization ability, security, stability, and reduced lock-in. Benefits provided by OSS and its unique development model must be balanced with a consideration of concerns such as technical knowledge of personnel, compatibility with existing software, problems with multiple operating system distributions, and documentation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

DISCLAIMER/DISCLOSURE

The research reported herein was performed as part of the Advanced Highway Maintenance and Construction Technology (AHMCT) Research Center, within the Department of Mechanical and Aeronautical Engineering at the University of California – Davis, and the Division of Research and Innovation at the California Department of Transportation. It is evolutionary and voluntary. It is a cooperative venture of local, State and Federal governments and universities.

The contents of this report reflect the views of the authors who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the State of California, the Federal Highway Administration, or the University of California. This report does not constitute a standard, specification, or regulation.

LIST OF ACRONYMS AND ABBREVIATIONS

Acronym	Definition
AASHTO	American Association of State and Highway Transportation Officials
ADIS	deprecated, see ATIS
ADUS	Archived Data User Services
AHMCT	Advanced Highway Maintenance and Construction Technology
AMTICS	Advanced Mobile Traffic Information and Communications System
APTA	American Public Transportation Association
APTS	Advanced Public Transportation System
ArcSDE	An ESRI Inc. proprietary mapping product
ARTCC	Air Route Traffic Control Center (FAA)
ARTS	Automated Radar Terminal System
ASOS	Automated Surface Observing System
ATC	Advanced Transportation Controller
ATIS	Advanced Traveler Information System
ATMS	Advanced Traffic Management System
AVCS	Advanced Vehicle Control System
AWOS	Automated Weather Observing System
BIOS	Basic Input Output System, a ROM chip
BSD	Berkeley Software Distribution
CACS	Japanese Comprehensive Automobile Traffic Control System
Caltrans	California State Department of Transportation
CCTV	Closed-Circuit TeleVision
CEO	Chief Executive Officer
CHP	California Highway Patrol
CIO	Chief Information Officer
CLI	Common Language Infrastructure
CMS	Changeable Message Sign
COPAMS	Commonwealth of Pennsylvania Air Monitoring System
COOP	NWS' Cooperative Observer Program
COTS	Commercial Off-The-Shelf Software
CVO	Commercial Vehicle Operations
CWOP	Citizen's Weather Observation Program
DEP	Department of Environmental Protection
DMS	Display Message Sign
DoD	Department of Defense
DRI	Caltrans' Division of Research and Innovation
DRIVE	Dedicated Road Infrastructures for Vehicle Safety in Europe
DTMC	Distributed Traffic Management Center
ECMA	European Computer Manufacturers Association
ERGS	Electronic Route Guidance System
ERTICO	European Road Transport Telematics Implementation Coordination Organization
ETMS	Enhanced Traffic Management System (FAA)
FAA	Federal Aviation Administration
FHWA	Federal Highway Administration

Acronym	Definition
FMCSA	Federal Motor Carrier Safety Administration
FOSS	Free and Open-Source Software
FSF	Free Software Foundation
FTA	Federal Transit Administration
GCC	GNU Compiler Collection
GDAL	Geospatial Data Abstraction Library
GIF	Graphics Interchange Format
GIS	Geographic Information System
GML	Geography Markup Language
GNU	The GNU Project (stands for GNU's Not Unix)
GPL	Free Software Foundation's General Public License
GRASS	Geographic Resources Analysis Support System
GST	Global System for Telematics
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDEA	Innovations Deserving Exploratory Analysis
IEC	International Engineering Consortium
IMAP	Internet Message Access Protocol
IOUG	Independent Oracle Users Group
IP	Internet Protocol
IRIS	Intelligent Roadway Information System
ISO	International Standards Organization
ISTEA	Intermodal Surface Transportation Efficiency Act
IT	Information Technology
ITE	Institute of Transportation Engineers
ITS	Intelligent Transportation Systems
ITS-4	4th generation ITS Technologies
IVHS	Intelligent Vehicle Highway System, deprecated, see ITS
J2EE	Java Platform, Enterprise Edition
JDK	Java Development Kit
JEMS	JBoss Enterprise Middleware System
JMS	Java Messaging Service
JNI	Java Native Interface
JPEG	Joint Photographic Experts Group, lossy image compression
JSP	Java Server Pages
JVM	Java Virtual Machine
KML	Keyhole Markup Language
LAMP	Software stack consisting of: Linux, Apache, MySQL, and Perl/PHP/Python
LEAPS	Lazy Evaluation Algorithm for Production Systems
MIT	Massachusetts Institute of Technology
MOTS	Modifiable Off-The-Shelf Software
MPEG-2/3/4	Moving Picture Experts Group codecs
MJPEG/M-JPEG	Motion JPEG video codec

Acronym	Definition
MVCC	Multi-Version Concurrency Control
NAS	National Academy of Sciences
NCRST	National Consortium on Remote Sensing in Transportation
NEMA	National Electrical Manufacturers Association
NEXRAD	NEXt-generation RADar, National Weather Service Doppler Radar
NFS	Network File System
NGSIM	FHWA's Next Generation Simulation project
NOAA	National Oceanic and Atmospheric Administration
NWS	National Weather Service
NTCIP	National Transportation Communications for ITS Protocol
OASIS	Organization for the Advancement of Structured Information Standards
ODF	OASIS Open Document Format
OGC	Open Geospatial Consortium
OGR	OSS library for reading vector files
OpenGIS	Open Geospatial Consortium
OpenSSH	Open Secure Shell
OpenSSL	Open Secure Sockets Layer
OS	Operating System
OSI	Open Source Initiative
OSS	Open Source Software
OSS/FS	Open-Source Software / Free Software
OWL	Web Ontology Language
PC	Personal Computer
PDA	Personal Digital Assistant
PHP	PHP Hypertext Preprocessor, a popular programming language
PPC	Power PC
PROMETHEUS	Program for European Traffic with Highest Efficiency and Unprecedented Safety
RACS	Road/Automobile Communication System
RAM	Random Access Memory
RDF	Resource Description Framework
RDF-S	Resource Description Framework Schema
RDP	Remote Desktop Protocol
RFC	Requests For Comments
RFQ	Requests For Quotation
RIITS	Regional Integration of Intelligent Transportation Systems (see pg. 48)
RISC	Reduced Instruction Set Computer
ROI	Return On Investment
ROM	Read Only Memory
RTD	Remote Traffic Detector
RTMC	Regional Transportation Management Center
RWIS	Remote Weather Information System / Roadway Weather Information System
RWS	Remote Weather Station
SAFE-T	Statewide Analysis For Engineering and Technology, Oklahoma application
SAFETEA-LU	Safe, Accountable, Flexible, Efficient Transportation Equity Act: A Legacy for Users

Acronym	Definition
SensorML	Sensor Model Language
SHP	ESRI Inc. vector shape file format
SOAP	Simple Object Access Protocol
SOS	Sensor Observation Service
SPARC	Scalable Processor ARChitecture, Sun Microsystems Inc.
SPARQL	SPARQL Protocol and RDF Query Language
SPS	Sensor Planning Service
SQL	Structured Query Language
SSL	Secure Sockets Layer
SVG	Scalable Vector Graphics
SWT	Standard Widget Toolkit
TCIP	Transit Communications Interface Profile
TCP/IP	Transmission Control Protocol/Internet Protocol
TEA-21	Transportation Equity Act for the 21st Century
TEXAS	Traffic EXperimental and Analysis Simulation
TMC	Traffic Management Center
TransducerML	Transducer Markup Language
TRB	Transportation Research Board
UCD	University of California Davis
USB	Universal Serial Bus
USDOT	U.S. Department of Transportation
VDOT	Virginia Department of Transportation
VERTIS	Japanese VEhicle, Road and Traffic Intelligence Society
VNC	Virtual Network Computing
W3C	World wide web consortium
WFS	Web Feature Service
WMS	Web Mapping Service
WRRHAMS	Weather-Related Road Hazards Assessment and Monitoring System
WSDL	Web Service Description Language
x86, x86-32	Commodity Intel, AMD, and compatible microprocessors
x64, x86-64	Commodity 64-bit Intel, AMD, and compatible microprocessors
XML	eXtensible Markup Language
XSLT	eXtensible Stylesheet Language and Transformation

ACKNOWLEDGEMENTS

The authors thank the California State Department of Transportation for their support; in particular, the guidance and review provided by the Open ATMS project team and Technical Advisory Group. The authors also acknowledge the dedicated efforts of the AHMCT development team members. Special thanks to Chad Bahrmann, Mohammed Bendelhoum, Alan Benson, David Cordone, Sean Coughlin, Jason Ellison, David Gibson, Mark Hallenbeck, Roya Hasas, Joe Havlicek, Robert Huck, James Kranig, Gene McHale, Bibiana McHugh, Jeff McRae, Dan Middleton, Val Noronha, Carlos Palau, Brian Park, Cesar Perez, Brian Simi, Stan Slavin, Brian Smith, Trey Tillander, and Fred Yazdan.

EXECUTIVE SUMMARY

Purpose

The purpose of this report is to summarize developments in Advanced Traffic Management System (ATMS) software and hardware as it relates to OSS (open-source software) and multi-source commodity servers based on the Intel x86 architecture. Case studies are discussed for ATMS and Intelligent Transportation System (ITS) projects that use OSS and commodity hardware. The authors recommend review of the cited references to gain a deeper understanding. Many of the cited references are available online, with links provided in the references. The authors also recommend using an Internet search engine to gain familiarity with new terms, concepts, and acronyms (see also pg. xv). The online encyclopedia Wikipedia may also be helpful (en.wikipedia.org).

What is Open-Source Software?

Open-source software (OSS)[1] is software that is distributed with its source code and is free (to all) or low-cost. Open-source software is enhanced and maintained by a community, an organized group of unpaid volunteers who contribute, maintain, and enhance it. Popular examples of OSS projects are the Linux operating system, the GNU (GNU's Not Unix) project, the MySQL database, and the Apache web server.[2] Open-source communities typically have defined rules on how they operate. For example, project leaders are responsible for deciding which features will be included in subsequent versions. OSS projects use a special type of copyright license that preserves every user's access to software source code.[3] The most popular open-source copyright license is the GPL (General Public License).[4] Open-source licenses ensures communal ownership of the source code and many open-source licenses protest communal ownership of derivative versions [61]. The key idea underlying OSS is that software is a kind of knowledge that is freely shared with and advanced by anyone interested in contributing [107]. In contrast, the traditional proprietary (retail) view of software source code is analogous to a manufacturing blueprint, and must therefore be protected and hidden from competitors. The ability to easily share source code and knowledge is a key attribute of the OSS approach, which produces a number of unique benefits.

The history of OSS is rich and has its roots in the creation of the ARPANET (the first packet-switched network), AT&T's Unix, Berkeley's BSD (Berkeley Software Distribution) Unix, the GNU project, and the Linux open-source operating system.[5] Netscape's decision in 1998 to open the source of their Internet browser was also an important event for the open-source approach.

---

[1] Throughout this report, we will use "open-source" where grammatically appropriate. However, it is important to note, e.g. for Internet searches, that many popular references use strictly "open source", i.e. with the hyphen omitted.
[2] See www.linux.org, www mysql.com, and www.apache.org.
[3] Appendix D (pg. 89) discusses OSS licenses in more detail.
[4] For a description of the GPL copyright license, see www.gnu.org/copyleft or en.wikipedia.org/wiki/GPL.
[5] For detailed accounts of the origins of OSS, see E.S. Raymond, *The Cathedral and the Bazaar* (available at www.catb.org/~esr/writings/cathedral-bazaar/), and G. Moody, *Rebel Code*.

Current Developments in Open-Source Software

OSS use is growing and shaping industry developments. The top three reasons cited for Linux use on servers are low cost (78%), reliability (74%), and performance (73%) (see Table 2, pg. 12) [7]. The top three reported concerns with Linux on servers are technical knowledge of staff (35%), compatibility with existing software (33%), and problems related to multiple versions of Linux (24%) (see Table 9, pg. 29). Among software developers, MySQL is approaching majority market share (44%), an increase of 25% between April and October of 2005 [40]. The Independent Oracle Users Group (IOUG) in a survey in early 2006 found that 44% of respondents will be running their Oracle databases on Linux in the next twelve months [106]. Linux is currently in use as a server operating system at 49% of companies polled [7]. The remainder are either pilot testing or will be in twelve months (23%), or have no plans (28%). Both IBM and Oracle certify and support their databases on Linux. Oracle offers their complete product line on Linux. Oracle has chosen to "bear hug" OSS rather than fight it, with Oracle's CEO, Larry Ellison saying "we are moving aggressively into open source. We are embracing it. We are not going to fight this trend. We think if we're clever, we can make it work to our advantage" [65]. Oracle has recently purchased OSS companies Sleepycat Software and Innobase with the strategy of generating revenue through service and support rather than initial or ongoing licensing fees. The OSS approach to software development is clearly having a large impact on the information technology market, and will continue to drive change.

How is Open-Source Software Different?

Sharing software source code increases efficiency by sharing costs and development risks with other interested organizations and individuals. However, organizations must be selective about which software they share. A useful distinction is between *differentiating* and *non-differentiating* software [87]. Differentiating software is software that distinguishes a company from its competitors for potential customers. In contrast, non-differentiating software is irrelevant to customer's evaluation of a company. For example, Excel is differentiating software for Microsoft's customers because it enhances (or differentiates) their view of Microsoft compared to competitors. The internal accounting software Microsoft uses is non-differentiating software for customers because it has no effect on how customers perceive Microsoft. This distinction is important because organizations have nothing to lose and much to gain by sharing (via OSS) their non-differentiating software. Sharing non-differentiating software increases efficiency because it enables spending additional resources on differentiating software, which enhances the organization's value to customers.

Comparing traditional retail closed-source (proprietary) software development with the OSS development model is beneficial. The OSS development model distributes development risks and costs among contributors. In contrast, proprietary software developers assume complete development risk and incur costs prior to shipping a product. New OSS projects with no collaborators are similar to the retail model. OSS tends to be more cost-efficient compared to retail development—every dollar spent is a dollar spent on software development. This contrasts with a company like Oracle, which spends about 13% of total revenue (the customer's money) on research and development.

A distinctive attribute of the OSS development model is a tendency to reduce customer *lock-in*. Lock-in is a term used in economics to describe a situation in which a customer or vendor faces high costs to switch products or technologies. OSS reduces lock-in in four ways. 1) There is no motivation for OSS contributors to introduce non-standard hooks to increase lock-in because there is no revenue motivation. 2) OSS contributors are inherently more interested in following and forming standards because this increases the desirability of the software to others, which encourages further contributions, more users, and increased benefits for everyone. The standards-following nature of OSS is an especially strong incentive for State, Federal, and local governments, which are often charged with following standards. 3) OSS markets are typically competitive and healthy with a large number of products from which customers can choose. 4) OSS projects tend to be portable because source code is available and projects are written for multiple platforms. This makes it relatively easier for customers to switch operating system platforms. Finally, by offering a free or low-cost substitute, an OSS product will tend to have a damping influence on proprietary product cost.

Another distinctive attribute of the OSS development model is how innovation is created. Generating innovation is a two-step process involving the creation of variations, followed by effective selection among these variations [61]. The OSS development process encourages innovation for a variety of reasons. The source code is available, which lowers barriers for worldwide personal participation. This tends to create a large and diverse group of contributors, for example academic researchers, students, hobbyists, government employees, etc. In addition, OSS projects tend to have a pervasive pragmatic attitude among group members. These factors and a complete lack of restrictions for contributors create a high degree of innovative variation. Effective selection among this variation is a competitive process based completely on what contributors, users, and developers find interesting and useful. New useful features and enhancements generate interest and attention, resulting in incorporation into subsequent versions. Because source code is available, innovations can more easily flow between different OSS projects. The innovative nature of the OSS development process is particularly suited for collaboration between government and research organizations.

What are Open-Source Software Strengths?

The OSS development model has a number of strengths, including reduced lock-in, reliability, security, development efficiency, a tendency towards healthy competitive markets, and a tendency towards the use of standards. This is a subset of benefits cited in other sources. In a report for the Department of Defense, Mitre Corp. identified eight key strengths: massive programming expertise, research and development covered by volunteer labor, an accepted leadership structure, quick release rates, parallel development and debugging, maturity of code, a culture of sharing, and long-term accessibility [63].

Increased reliability is often cited by OSS users as a primary benefit. It has been estimated that software defects cost the U.S. economy $59 billion annually [2]. The OSS development process creates software that tends to have fewer defects, with discovered defects repaired faster [89]. The basis of the OSS development process (see Section 4) is peer review of code and the ability of anyone to find defects and forward patches to developers. OSS communities have developed effective methods to test and validate patches via a large and active developer and user base (a key factor, as noted herein). The effectiveness of peer-reviewed development has

been quantified in a three-year $1.24 million study for the Department of Homeland Security called the "Vulnerability Discovery and Remediation Open Source Hardening Project" [79]. This program is part of a larger Federal effort to perform security audits of approximately 40 open-source software packages such as Linux, MySQL, and Apache. The project uses automated source code scanning tools (from Coverity Inc.) which are based on research by Stanford University. The mean number of defects per 1000 lines of source code in 32 open-source projects was found to be 0.434 [38]. MySQL was found to have .224 defects per 1000 lines of code, which is four times better than typical commercial software. Defect rates for popular OSS projects are shown in Table 7 (pg. 24). There is an inverse relationship between the number of users an OSS project has and the defect rate. The OSS development process also produced rapid repairs of discovered defects [50].

Increased security is another primary benefit often cited by OSS users. When considering the security of any system, healthy skepticism is in order—ultimately security depends on knowledgeable, skilled, and paranoid system administrators. A qualitative approach is one way of evaluating security. Using this approach indicates a clear security preference for OSS among system administrators and Chief Information Officers (CIOs) over proprietary software [63]. However, there is healthy skepticism whether disclosing source code makes the system more secure [109], while at the same time there is healthy skepticism that not disclosing source code provides too great of a temptation for proprietary software vendors to hide numerous and critical security vulnerabilities. Quantitatively, OSS security advisories are repaired faster. The corporate security provider Secunia Inc. tracks and reports security advisories for a number of OSS and proprietary software products [27].

Table 8 (pg. 26) shows the number of unpatched and patched security advisories for a number of products. The data shows that the OSS products were rapidly repaired following discovery of vulnerabilities. At the time of this writing, the LAMP (Linux, Apache, MySQL, and PHP) stack, consisting of Red Hat ES 5/SUSE, Apache, MySQL, and PHP, contained two unpatched advisories. By comparison, over a five-month period Microsoft's Windows XP reduced its number of unpatched advisories by one, down to 28. Some of these advisories are rated as "highly critical." The quantitatively lower defect rate and faster repair rate of OSS is strong objective evidence of superior security.

Increased efficiency is another strength of the OSS development model. For developers, the availability of existing software code that can be enhanced improves efficiency. The accessibility of OSS over the Internet through a simple download is also powerful. No purchase requisition forms are required—this can be a significant time and cost savings in an organization. An individual with time, skill, and an Internet connection can solve previously unsolvable problems. Updates can be provided automatically using the base operating system's update manager capabilities. On a State or National level, one can imagine the benefits from a National ATMS open-source effort in which individual State DOTs contribute their best functionality or components. Potentially, each DOT could gain from the best that every other state could contribute, a net gain for all, including the traveling public.

A healthy ITS market is one of the six primary goals of the National ITS program [6]. By definition, a market includes both buyers and sellers [98]. A healthy market results when the benefit for both buyers and sellers is maximized over the long run. The use of OSS in ATMS

contributes to a healthy ITS market in a number of ways. First, there are no (or small) software acquisition costs, which frees funds for more productive uses, such as adding features. Second, the use of OSS products facilitates cooperation and development between cost-constrained universities and DOTs. Third, the use of OSS enables contributions, such as enhancements and added features, to OSS projects from which other users (e.g. other DOTs) benefit. Fourth, DOTs would experience less software lock-in (see Section 4, pg. 15).

The use and development of ATMS standards is crucial for the healthy development of the ATMS market and interoperability (see ITS Standards in Section 7). Because OSS is intrinsically about sharing knowledge, there is inherent motivation to use and develop standards. Any DOT, consultant, or university researcher contributing to an OSS project would want their contributions to be included in subsequent versions of the product. Contributions that do not conform to existing standards are much less likely to be used by others and less likely to be included. Further, potential enhancements to OSS projects that ignore standards tend to increase lock-in, which users avoid and view as a dead-end (see "Lock-in" in Section 4).

What are Open-Source Software Concerns?

Any new or disruptive technology has a number of concerns. Key concerns about OSS include lack of trained staff, inconsistent quality, and support. Other concerns have been reported elsewhere. For example, Mitre reports OSS weaknesses as: lack of ownership, difficulty in starting new OSS projects, and less user-friendliness [63]. Table 7 (pg. 29) shows top issues reported in a 2005 study of Linux deployment on servers [7]. Deployment issues that are addressable by commercial markets can be expected to improve. For example, a key concern in 2002 was a lack of vendor support, which was addressed by the market by 2005.

A sufficient number of trained staff can be a problem if an organization has no Unix or Linux experience. Nearly all college graduates in computer science or engineering have some amount of experience with Linux. This trend is expected to improve. Training for existing staff is available from a number of vendors. The FAA used this approach in their conversion of the Enhanced Traffic Management System to OSS (see the case study, pg. 44). Sticking with mainstream OSS applications (e.g. MySQL, Eclipse, PHP, etc.) increases the pool of potentially available trained staff, as well as available support through online forums.

Inconsistent quality is a concern for new OSS projects, or projects with fewer users and developers. There is a clear relationship between project code quality and the number of users. This also carries over to support, documentation, and security concerns. New OSS projects also face an origination problem—it is difficult to attract users and developers to small projects.

Support is often crucial for project success. The availability of support has dramatically improved since 2002 when it was a top issue. Both paid support and free support (Table 8, pg. 30) are available for hardware, Linux, and mainstream applications such as MySQL. A unique feature of the paid open-source support market is its competitive nature, which results from the availability of source code, lowering entry barriers for new firms. Of particular importance to California, Oracle's support for Linux is extensive—all Oracle products are available on Linux with full support.

What is ATMS?

The field of Advanced Traffic Management Systems (ATMS) is a primary subfield within the Intelligent Transportation System (ITS) world. The ATMS view is a top-down management perspective that integrates technology primarily to improve the flow of vehicle traffic and improve safety. Real-time traffic data from cameras, speed sensors, etc. flows into a Transportation Management Center (TMC) where it is integrated and processed (e.g. for incident detection), and may result in actions taken (e.g. traffic routing, CMS messages) with the goal of improving traffic flow. The National ITS Architecture defines the following primary goals and metrics for ITS [6]:

- Increase transportation system efficiency,
- Enhance mobility,
- Improve safety,
- Reduce fuel consumption and environmental cost,
- Increase economic productivity, and
- Create an environment for an ITS market.

ATMS History

In 1956, the *National Interstate and Defense Highways Act* initiated a 35-year $114 billion program that designed and constructed the Interstate highway system. This hugely successful program was mostly complete by 1991, and the era of build-out was over. In the mid to late 1980s transportation officials from Federal and State governments, the private sector, and universities began a series of informal meetings discussing the future of transportation. This included meetings held by the California State Department of Transportation (Caltrans) in October of 1986 to discuss technology applied to future advanced highways [112]. In June of 1988 in Washington, DC, the group formalized its structure and chose the name Mobility 2000 [93]. In 1990, Mobility 2000 morphed into ITS America, the main ITS advocacy and policy group in the US. The initial name of ITS America was IVHS America and was changed in 1994 to reflect a broader intermodal perspective [111].

The 1991 *Intermodal Surface Transportation Efficiency Act* (ISTEA) was the first post-build-out transportation act. It initiated a new approach focused on efficiency, intelligence, and intermodalism. It had a primary goal of providing "the foundation for the nation to compete in the global economy" [60]. This new mixture of infrastructure and technology was identified as an *Intelligent Transportation System* (ITS) and was the centerpiece of the 1991 ISTEA act. ITS is loosely defined as "the application of computers, communications, and sensor technology to surface transportation" [59]. Subsequent surface transportation bills have continued ITS funding and development. In 2005 the SAFETEA-LU (Safe, Accountable, Flexible, Efficient Transportation Equity Act: A Legacy for Users) surface transportation spending bill was signed into law.[6] It continues healthy ITS implementation and research funding.

[6] See www.fhwa.dot.gov/safetealu.

ITS Standards

National nonproprietary standards are crucial for the healthy development of the ITS market [94]. Standards define what information is open and shared for participants. To meet ITS standardization needs the U.S. Department of Transportation initiated development of the *National ITS Architecture* in 1994. The National ITS Architecture provides a common framework for ITS development by defining systems and subsystems and the data that flows between them. The 2005 SAFETEA-LU surface transportation bill continues strong support of the National ITS Architecture: "to the maximum extent practicable, the national architecture shall promote interoperability among, and efficiency of, intelligent transportation system technologies implemented throughout the United States."[7] In 2001, the TEA-21 act stipulated denial of Federal funding for any ITS project that does not conform to the National ITS Architecture. SAFETEA-LU continues this requirement.[8]

ATMS Trends

Trends in ATMS and ITS are driven both internally by customer needs and externally by advances in technology. Semiconductor process technology continues to lower power requirements, reduce chip size, and double performance about every 18 months. This is simultaneously pushing decentralization of processing intelligence and commoditization of sensors, processors, and communication. These are combined to produce results in real time, and have been described as next-generation "ITS-4" technologies [96]. The commoditization of hardware parallels the commoditization of software in the form of OSS. For example, the Advanced Transportation Controller (ATC) standardization effort is taking advantage of these trends by using commodity hardware and OSS (case study, pg. 63). Commodity IP video cameras are another example and are referenced in several case studies.

Trends inside the ATMS market are continuing to push change. As ITS build-out continues into lower-density population areas, cost-sensitivity increases—the primary obstacle to implementing ATMS is no longer technology, it is cost. This is exemplified by California's hesitation to implement its existing ATMS system in all twelve districts. Other trends are the increased use of standards and increasing ITS interoperability requirements. Finally, increasing Homeland Security requirements appear to be a long-term trend driving some ATMS functional requirements such as video surveillance.

Case Studies of ITS Software Projects Using Open-Source Software

This section discusses case studies of closed-source ITS software projects that *use* open-source software, which are distinct from projects that *are* open-source projects. For example, the projects might *use* MySQL, Linux, Apache, etc. along with developed software code to build the application. The developed source code is *not* shared with other organizations. The focus here is on ATMS projects; however, other related projects are discussed. It is significant that many of the projects listed here are explicitly low-cost and many are associated with research organizations. This is significant because research organizations tend to be more cost-constrained

[7] See www.fhwa.dot.gov/safetealu, section 5307, subsection A.
[8] See the SAFETEA-LU Public Law, www fhwa.dot.gov/safetealu, section 5307, subsection C.

than State DOTs. It is clear that the increased use of OSS by State DOTs will increase the flow of innovation, ideas, and new developments between DOTs and research institutions, benefiting both, and the traveling public.

The Virginia Web-based Congestion Monitoring ATMS project is implemented using OSS by the Smart Travel Lab at the University of Virginia [95]. The project was completed in September of 2003. It uses an existing Oracle database for historical traffic data, and the OSS database MySQL for real-time data. MySQL was used to lower the load on the existing Oracle database and to reduce Oracle licensing fees [86]. MySQL was also used because it was "free, reliable, easy to use, [and] has [a] large user base" [85].

The Oklahoma Statewide Distributed Low-cost ITS project is implemented using OSS. It uses a novel distributed architecture. It is being developed by the University of Oklahoma [57]. Cost was a major concern, so a distributed ITS was designed to keep costs low. The guiding design philosophy is that "any console should be able to control any system resource at any time." This enables the elimination of an expensive centralized TMC. Operators are connected by a peer–to-peer network into a virtual, geographically distributed, and fault-tolerant TMC. The ITS console runs on commodity x86 hardware, Microsoft Windows XP, and the following OSS products: Apache, PHP, MySQL, and the MapServer GIS.[9]

The Oklahoma SAFE-T (Statewide Analysis For Engineering and Technology) Accident Analysis System is a project that uses OSS [99]. The Oklahoma Department of Transportation is funding the development for traffic engineering decision-support [99]. It is presently being used by the Oklahoma DOT, the Oklahoma Highway Patrol, and municipal and local traffic engineers statewide. The goal is to reduce crashes using automated traffic analysis of highway enhancements and construction. The system uses the OSS MySQL database and MapServer GIS products [56,58]. The University of Oklahoma is developing the system.

The Oklahoma ATIS (Advanced Traveler Information System) System is a project that uses OSS and is being developed by the University of Oklahoma [103]. It will provide real-time traffic data for Oklahoma City and will be used by TMC personnel and the traveling public. The system will provide real-time average speed, road conditions, construction information, and web-cam images. The system is implemented with the OSS database MySQL, the Apache web server, and the MapServer GIS [56].

The Minnesota IRIS Intelligent Roadway Information System (IRIS) is an ATMS project that uses OSS and was developed by the Regional Transportation Management Center (RTMC) within the Minnesota DOT (Mn/DOT). IRIS provides ramp metering, incident detection, and CMS control. It uses Linux and the OSS database PostgreSQL[10], and is implemented with Java. The client displays a GUI road map and is supported on Linux and Windows. Commodity x86 hardware is used, including PCI-based serial communication with field elements. Mn/DOT's RTMC implements all of their software projects using OSS and is interested in collaborative partnerships with other DOTs or organizations. Other RTMC projects are using Apache, Tomcat,

[9] See mapserver.gis.umn.edu.
[10] See www.postgresql.org.

Batik, Hibernate, Struts, Velocity, Ant, Eclipse, Python, and Mercurial.[11] Reported benefits of their OSS approach are low initial and recurring costs, and the ability to customize the software. Software stability and reliability are also reported to be high.

A relevant case study is the Federal Aviation Administration's (FAA) recent technical refresh of their Enhanced Traffic Management System (ETMS), which offers insight into what is possible using commodity x86 hardware and OSS. During the technical refresh in 2004 and 2005, the FAA replaced all proprietary RISC (Reduced Instruction Set Computer) workstations with commodity x86 workstations running Linux, reducing costs from $25,000 to $3,000 per workstation. The ETMS system has 1,200 workstations scattered across 100 sites. Also, 1.5 million lines of software code were ported to Linux [16]. The initial cost estimate for the technical refresh of the original proprietary RISC system was $25 million; however, the project was achieved with $10 million. In addition, performance improved substantially [13].

The Weather-Related Road Hazards Assessment and Monitoring System (WRRHAMS) is a project that uses OSS and was funded by the National Consortium on Remote Sensing in Transportation (NCRST) [20]. The project goal was to develop a web-based application "for identifying and visualizing transportation infrastructure locations in danger of having been damaged by precipitation events" for rural unpaved roads in New Mexico [114]. The system was implemented using Linux and the OSS GIS GRASS (Geographic Resources Analysis Support System).[12]

The Pennsylvania Hourly Mesonet, which also uses OSS, is a centralized repository of historic and real-time weather station data maintained by the Pennsylvania State Climatologist[13]. Hourly data is received from approximately 160 networked weather stations (Figure 18) across Pennsylvania which report temperature (Figure 19), wind direction, wind speed, and other information [102]. Processed data includes wind streamlines, temperatures, dew-point contours, and associated time series. The project used the OSS Perl scripting language, Apache web server, MySQL database, and Red Hat Enterprise Linux operating system.

The Los Angeles County RIITS (Regional Integration of Intelligent Transportation Systems) project is implemented with OSS and has a primary goal of increasing the effectiveness of investments already made in electronic information systems and transportation systems through the real-time exchange of traffic and transportation data. Agencies involved are Caltrans, the California Highway Patrol, the City of Los Angeles Department of Transportation, and the Los Angeles County Metropolitan Transportation Authority. RIITS uses an innovate approach with extensive use of web services and XML for data integration. OSS used includes Red Hat Fedora Linux, JDK 1.4 (Java Development Kit), Apache Tomcat JSP (Java Server Pages) Server, and the Apache Axis SOAP provider [70].

The University of Maryland CATT Laboratory (Center for Advanced Transportation Technology) is developing RITIS (Regional Integrated Transportation Information System)

[11] See www.apache.org, tomcat.apache.org, xmlgraphics.apache.org/batik, www hibernate.org, struts.apache.org, jakarta.apache.org/velocity, ant.apache.org, www.eclipse.org, www.python.org, www.selenic.com/mercurial.
[12] See grass.itc.it.
[13] See the Pennsylvania Mesonet site at pasc.met.psu.edu/MESONET.

using open-source software14. The goal of RITIS is to improve transportation efficiency, safety, and security by integrating existing real-time and archival regional transportation data from Virginia, Maryland, and the District of Columbia, among others (see Figure 21). RITIS is built using Apache, PHP, MapServer, PostgreSQL, the JBoss Application Server, MySQL, JDOM, and Log4J. The CATT Laboratory makes extensive use of OSS.

The University of Maryland CATT Laboratory is developing an incident management system using open-source software for the Virginia Department of Transportation (VDOT). This incident management system was design and developed rapidly to meet VDOT's immediate TMC software needs until a full featured solution is developed. It was developed using the Apache HTTP Server, PHP, MapServer, PostgreSQL, AMFPHP, and the Wildfire Server.

Case Studies of ITS Open-Source Software Projects

This section discusses case studies of ITS software projects that *are* open-source software projects. The prior section covered closed-source software projects that *use* open-source software. Projects discussed here are explicitly organized to share project artifacts: source code, datasets, test results, etc., with a community of users who may contribute to the project in some form. The community benefits from these contributions. These projects use an OSS license to grant users access to project artifacts. See Appendix D for information on open-source licenses (pg. 89). The focus here is on ATMS projects; however, other related projects are discussed.

The MIT Open-Source MITSIMLab Traffic Simulator is an open-source project that is implemented using OSS. It is developed by MIT's Intelligent Transportation Systems Program and is used to evaluate ATMS and ATIS systems [75]. MITSIMLab uses an open-source license that requires modifications and enhancements to be posted in the public domain. Support for the open-source version is provided by a public newsgroup that has 45 members as of May 2006.

The Federal Highway Administration's Next Generation Simulation program (NGSIM) uses the open-source approach by releasing some project products using an open-source license and encouraging community involvement. It is openly sharing data sets, documentation, and algorithms with the transportation community. Program goals are to improve traffic simulation tools, promote the use of simulation, and ensure the accuracy and trust of traffic simulation tools by providing validated simulation results. The NGSIM program uses the Creative Commons copyright license,[15] which is similar to the GNU GPL license. It grants users redistribution rights and requires attribution.

The TEXAS (Traffic EXperimental and Analysis Simulation) intersection simulation model is an open-source project. In May of 2003 the United States Department of Transportation (USDOT) requested enhancements to the TEXAS microscopic single intersection simulation model [3]. The TEXAS simulator models sub-microscopic behavior of vehicles as they pass through intersections and mix with other traffic flows. The source code for this project is licensed using the Free Software Foundation's (FSF) open-source GPL license.

[14]For the University of Maryland CATT Laboratory, see www.cattlab.umd.edu.
[15] See Appendix D (pg. 89) for more information and creativecommons.org.

The University of Washington Urban Simulation and Modeling Project is an open-source project. UrbanSim is developed by the Center for Urban Simulation and Policy Analysis at the University of Washington, Seattle[16]. UrbanSim models economic relationships between actors such as households, businesses, developers, and government policies. UrbanSim uses the OSS projects Python, MySQL, and GDAL. It has been tested with Windows XP, SUSE Linux, and Mac OS X. UrbanSim uses the GPL open-source license.

The Dynamic Timetable Generator project is a prototype proof-of-concept open-source project from the TRB IDEA (Innovations Deserving Exploratory Analysis) program. The goal of the project is to provide a general-purpose open-source tool that dynamically generates transit timetables for customers accessing a transit web site. Developed software code will be licensed using an open-source license.

The TimeTable Publisher is an open-source project of the Oregon Tri-County Metropolitan Transportation District (TriMet). The goal of TimeTable Publisher is to provide a general-purpose open-source tool that dynamically generates transit timetables for customers accessing a transit web site. TimeTable Publisher is based on some ideas from TRB's prototype IDEA project, and has benefited from a number of lessons learned during the prototype. TriMet will release TimeTable Publisher using an open-source license. TriMet uses OSS extensively and reports a number of positive technical and organizational benefits.

Case Studies of ITS Hardware Projects Using Open-Source Software

This section discusses case studies of ITS hardware projects that use OSS in some way. The focus is on ATMS-related projects; however, other transportation-related projects are discussed.

Peek Traffic Inc. sells a Linux-based automatic traffic data recorder, the ADR-6000 [24]. It is a rack-mount unit with a Pentium-class processor. The Texas Transportation Institute performed an evaluation of the ADR-6000 and three similar units [44]. The researchers noted that the unit was comparable to the others and fell near the bottom of the price range.

The city of Valencia Spain recently deployed a traffic management system that uses OSS for video streaming over TCP/IP [49]. The guiding philosophy used to build the system was to use Commercial-Off-the-Shelf (COTS) hardware and OSS components that follow open standards and avoid proprietary systems. TCP/IP video cameras using MPEG-4 were used, instead of traditional closed-circuit TV (CCTV), to provide increased scalability, lower cabling costs, and facilitate providing video streams to the public over the Internet. The system uses OSS: Linux, MySQL, the video player VideoLAN Client (VLC), and the multimedia system FFMPEG[17], among others [83]. A diagram of the main system components is shown in Figure 31, pg. 64.

Case Studies of ITS Open-Source Hardware Projects

This section discusses case studies of ITS hardware projects that *are* open-source projects—they attempt to build a community of users sharing software code, experience, improvements,

[16] For UrbanSim, see cuspa.washington.edu.
[17] See www mysql.com, www.videolan.org, and ffmpeg mplayerhq hu.

etc. The community benefits from these contributions. Projects in this category use some type of OSS license to grant users access to project artifacts. See Appendix D for information on open-source licenses (pg. 89).

The Advanced Transportation Controller (ATC) project uses OSS and is an OSS project. It is a standardization effort initiated by the USDOT, and sponsored by the American Association of State and Highway Transportation Officials (AASHTO), the Institute of Transportation Engineers (ITE), and the National Electrical Manufacturers Association (NEMA). It has been recognized as one of the most important emerging standards for the ITS community. The ATC will provide an open software and hardware platform for a wide variety of ITS applications, and is viewed as similar to a ruggedized field-deployed personal computer. Table 10 (pg. 63) lists some anticipated applications. The ATC standard specifies the Linux operating system. The ATC standardization effort is needed for several reasons. First, there is a growing need to integrate controllers that were previously incompatible. Second, increasingly sophisticated ITS applications require supporting a broader set of applications, faster communication, and the ability to evolve over time. The ATC standard identifies these as broad ITS trends that are pushing device intelligence closer to the field, and driving the need for field devices that can run more sophisticated applications.

The European Global System for Telematics Open System project both uses OSS and is an OSS project. ERTICO is the European equivalent to ITS America, and sets standards, funds research, and coordinates projects. An ongoing project is the Global System for Telematics (GST), which has the goal of facilitating the creation of an open market for in-vehicle ITS services [15]. The GST Open Systems sub-project is delivering an open telematics framework consisting of the specifications, architecture, and a reference implementation for in-vehicle applications. The open telematics framework will be used by service providers, car manufacturers, and mobile end-users. The project uses open standards and source, including the JBoss J2EE application server, the SyncML synchronization standard, HTTP, Simple Object Access Protocol (SOAP), TCP/IP, and the Eclipse development environment, among others. See Figure 32 (pg. 65) for more information.

Conclusions

The ATMS and ITS markets are being driven both internally by customer needs and externally by advances in technology (shown simplified in Figure 1 below, and in Figure 34, pg. 70). Some of the trend drivers are:

- Improvements in semiconductor process technology pushing commoditization of sensors, processors, communication, and decentralization of processing intelligence, providing the capability for real-time decision-making (also called “ITS-4” technologies).
- Needs for lower-cost ATMS due to build-out into lower-density population areas.
- Ongoing needs to reduce congestion, lower accident rates, and increase mobility.
- Homeland Security requirements.

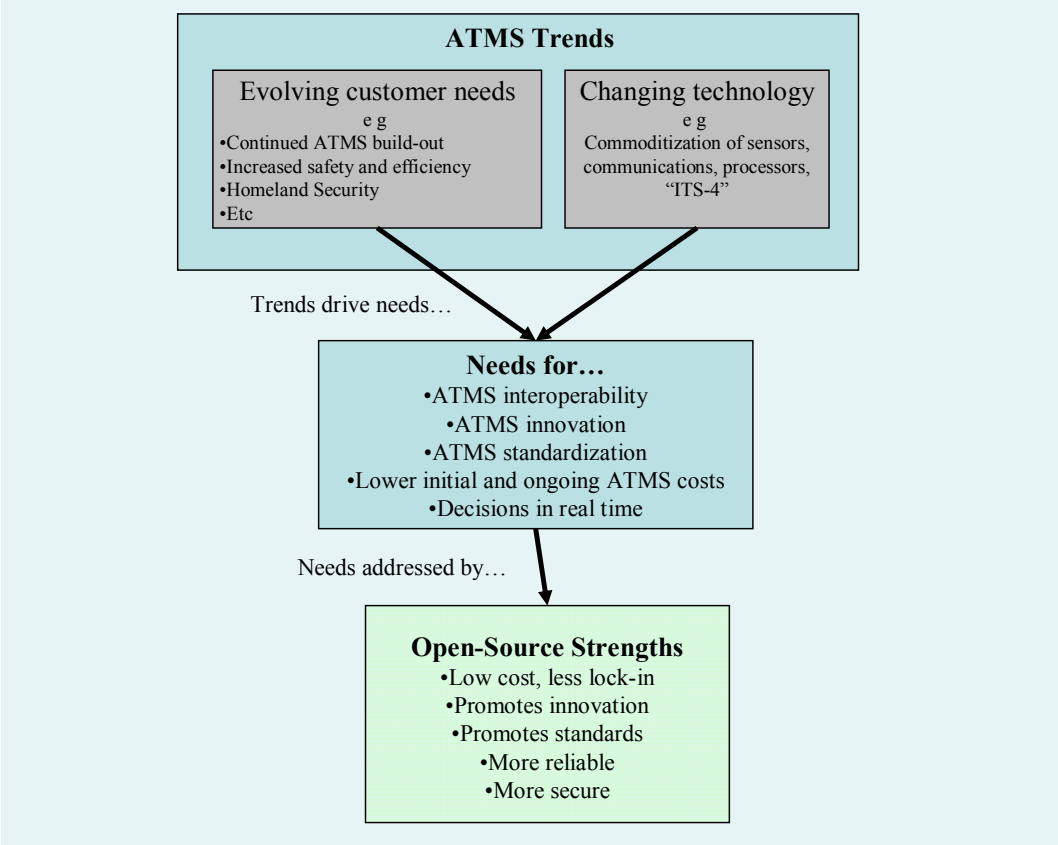

Figure 1: ATMS trends, needs, solutions (simplified)

These trends are driving needs for further innovation, better ATMS interoperability, more and better real-time data, and the ability to make real-time traffic management decisions. These needs are driving the necessity for software that is less expensive, but more secure, reliable, standards-based, and innovative.

Improvements in safety, congestion, and efficiency have been achieved with ITS applications and proprietary software. This report details the use of OSS to build ITS and ATMS applications. Each of the ITS projects covered in the case studies used OSS in one of two ways. The first type of project case study used open-source software to implement a closed-source application—ten of the case studies used this approach. Project organization is more or less unchanged compared to projects using proprietary products. The second type of project used open-source software to implement an open-source application—five of the case studies used this approach. These projects are organized to share project source code, datasets, test results, etc. with a community of users who may contribute to the project in some form. The community as a whole benefits from these contributions. Projects in this category use some type of OSS license to give a community ongoing access to project artifacts.

Benefits provided by OSS and its unique development model must be balanced with a consideration of some of the concerns, including lack of trained staff, less user-friendliness, and inconsistent quality. An understanding of the OSS development model is crucial for setting expectations. For example, benefits gained from peer review are proportional to the number of

individuals involved. As shown in the case studies, measured use of OSS with a consideration of OSS strengths and concerns can provide immense benefits to an organization.

There are three conclusions reached here. First, a number of ATMS and ITS projects have constructed applications using open-source software. For example, Minnesota's IRIS ATMS, Oklahoma's Statewide distributed ITS, and the National ATC standard for field devices all use OSS. In addition, others have used OSS to implement large and complex non-ITS applications. Examples include the FAA's technical refresh of the Nation's Real-time Enhanced Traffic Management System, and a number of corporations that run their businesses on OSS and commodity hardware such as Google, E*TRADE, Sabre, Travelocity, Yahoo!, and Amazon.

Second, as summarized in Figure 1, some of the benefits reported in the case studies related to the OSS development model parallel trends in ATMS and ITS. For example, high deployment costs are a concern for continued ATMS build-out in California. The need for increasingly sophisticated software applications in field devices and TMCs are also driving higher software costs. These cost concerns parallel OSS cost benefits reported in some of the case studies (e.g. see the Mn/DOT IRIS System, pg. 43). Reduced lock-in is closely associated with zero or low software licensing costs. Needs for higher ATMS reliability and security parallel reliability and security strengths reported with the OSS model, which may benefit ATMS applications.

Third, in the ITS field in general, there appears to be increasing interest in collaboration. This parallels the general requirement that ATMS and ITS systems increase functional interoperability. One-third of the cited case studies (5) are themselves open-source projects using open-source licenses to ensure community access to artifacts. A number of individuals contacted for this report and involved in the case studies expressed interest in sharing their proprietary work with others in the form of open-source ATMS projects. The open-source development model facilitates collaboration, the exchange of ideas, software code, and pooled funding which spreads development risks. DOTs, cities, and counties may benefit from continuing this collaborative approach. Collaborative access to source code may also increase competition among prospective firms because of lowered entry barriers. Firms or individuals that did enhance an OSS ATMS could not claim a proprietary right to the source code they wrote, potentially reducing lock-in for State DOTs. However, adopted enhancements would gain them visibility and recognition, which would increase their ability to win contracts for future enhancements.

SECTION 1:
INTRODUCTION

The subjects of this report are Advanced Traffic Management Systems (ATMS), open-source software (OSS), and the relationship between them. This report is the accumulated effort of a task within a larger multi-year research study undertaken by the Advanced Highway Maintenance and Construction Technology (AHMCT) Research Center at the University of California, Davis (UCD). The project title is *Research & Development of an Open-Source Advanced Traffic Management System*, with funding by the California State Department of Transportation (Caltrans). The report objectives are to:

- Summarize the history and current developments in ATMS software and hardware as it relates to OSS and commodity x86 hardware,
- Summarize the history, strengths, and weaknesses of OSS, and
- Summarize relevant ATMS hardware and software projects that use OSS, or are OSS projects themselves.

This report assumes light familiarity with ATMS and Intelligent Transportation System (ITS) concepts and less familiarity with software concepts. A brief introduction is provided to disruptive technology and product cycles. Sources of information are research journals, reports, news sources, web sites, and communication with individuals from State DOTs and research institutions. The authors recommend review of the cited references to gain a deeper understanding. Many of the cited references are available online, with links provided in the references. The authors also recommend using an Internet search engine to gain familiarity with new terms, concepts, and acronyms (see pg. xv). The online encyclopedia Wikipedia may also be helpful (en.wikipedia.org).

Improvements that have been achieved in safety, congestion, and efficiency with ATMS and proprietary software and hardware are impressive. This report implicitly asks the question: is it reasonable to use OSS and commodity hardware to build ATMS applications to achieve *further* benefits? Based on the information collected and analyzed for this report, the answer is yes. In addition, it appears that unique benefits may be derived from the open-source development model, such as increased innovation and collaboration among DOTs and research institutions, enhanced ATMS market health, and increased interoperability.

Following this introduction, Section 2 discusses how and why technology markets change, the nature of innovations, product lifecycles, disruptive innovation, and the commoditization of the computer hardware market. Section 3 discusses OSS history and recent developments. Sections 4-6 discuss how OSS development is different, software engineering, types of software development, and OSS strengths and weaknesses. Section 7 discusses ATMS history, goals, and trends. Finally, sections 8-11 discuss case studies of OSS use in ATMS and ITS hardware and software projects.

SECTION 2:
HOW AND WHY TECHNOLOGY MARKETS CHANGE

The computer hardware market is undergoing intense long-term change. Expensive RISC (Reduced Instruction Set Computer) systems from Sun, IBM, and Hewlett-Packard are being challenged. This raises a number of interesting general questions. What drives change? Can companies adapt to changing markets? What are the customer benefits and risks of adopting newer technology? This section attempts to address these questions, and is important for understanding change in any technology market. Key conclusions are that the RISC/Unix hardware market is being severely challenged by a disruptive technology: commodity hardware and OSS. OSS also appears to have the traits of a disruptive technology for existing software vendors.

The RISC/Unix market is less than 40 years old and relatively young. Another much older market is faced with change from commodity hardware and open-source—the encyclopedia market. The venerable Encyclopædia Britannica[18] was first published in 1768. Britannica has 238 years of experience, 4,000 contributors, and approximately 120,000 articles in the online version [14]. Compare this with Wikipedia, a free, online, and open-source encyclopedia that anyone can easily edit. It has existed since 2001 and contains over one million entries in 200 languages [115]. Figure 2 shows Wikipedia's growth rate. What about quality? There have been complaints about accuracy, completeness, etc.[19] However, the journal *Nature* performed a comparative review of encyclopedia accuracy and found that "among 42 entries tested, the difference in accuracy was not particularly great: the average science entry in Wikipedia contained around four inaccuracies; Britannica, about three" [55]. To emphasize, Wikipedia started in 2001, is free, has eight times the number of entries, is built by volunteers, is growing exponentially[20], and shows no signs of slowing. The good people of Britannica surely must be asking themselves: what is happening? How is it possible their existence could be threatened by a *free* encyclopedia?

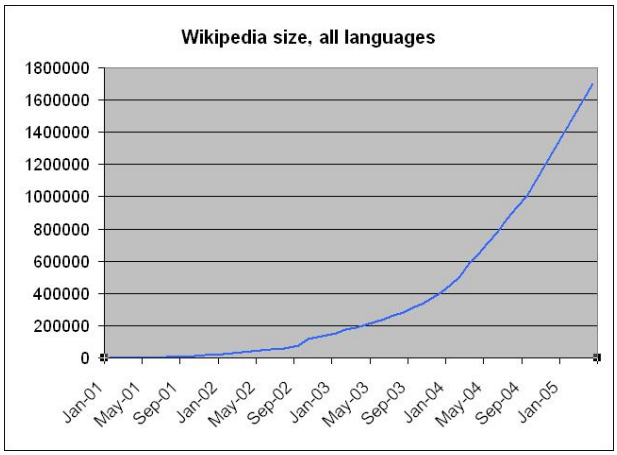

Figure 2: Total Wikipedia Encyclopedia articles in all languages [115]

[18] All products referenced in this report are registered, copyright, or trademark of their respective owners. These marks are omitted throughout for conciseness.

[19] For a list of complaints about Wikipedia quality, see: en.wikipedia.org/wiki/Wikipedia_Sucks.

[20] As of June 2006, Wikipedia is adding 62.8 new articles and 143.5 new users *per hour*. See www.wikiside.com.

Innovations Are Sustaining or Disruptive

Open and competitive markets as diverse as encyclopedias, hard disks, accounting software, mechanical excavators, and ATMS share a number of interesting traits. Exploring these traits is worthwhile and important for understanding the present and future of ATMS, OSS, and the relationship between them. To this end, Christensen's concepts of product lifecycles, disruptive innovations, and sustaining innovations are very helpful [39].

Consider any open market with competition. For example, motorcycles, disk drives, or fishing reels. These markets can be characterized by a stream of innovations. Any innovation can be characterized as *sustaining* or *disruptive*. Sustaining innovations extend the performance of existing products using metrics customers already use to evaluate the products. For example, adding additional cores to a microprocessor extends performance, which is a metric the customer already uses to evaluate processors. Other sustaining examples are larger LCD monitors, word processors with more features, and larger hard drives.

In contrast, disruptive innovations result in a change in the customer's primary product evaluation criteria. Familiar disruptive examples include the invention of the inexpensive microprocessor (price was the new metric), a reduction in the physical size of hard drives (e.g. 1-inch drives used in cameras), and the bundling of previously separate productivity software applications into the Office Suite by Microsoft (the new metric was convenience). Table 1 shows other sustaining and disruptive examples.

Table 1: Sustaining and disruptive products [39]

Sustaining Technology	**Disruptive Technology** (relative to sustaining technology)
Processors built with logic boards	Microprocessor
Traditional accounting software	Quicken QuickBooks
Large motorcycles, e.g. BMW, Harley-Davidson	Honda 50cc Supercub (1960)
Circuit-switched networks	Packet-switched networks
Traditional retailing	Internet-based retailing
Integrated steel mill	Mini-mill
Department stores	Discount retailing, e.g. Kmart, Wal-Mart
Manned fighters and bombers	Pilotless drones
Open surgery	Arthroscopic and endoscopic surgery
Minicomputers	Microcomputers
Vacuum tubes	Transistors

Characteristics of Disruptive Products

Products based on disruptive innovations tend to have a number of unusual characteristics that distinguish them from existing products. Disruptive products tend to be simpler, less expensive, more reliable, and improving at a faster rate than traditional products. Because of these properties, they are especially attractive to customers in new markets. For example, Hewlett-Packard's 1.3-in Kittyhawk drive (Figure 3) was unexpectedly used by digital camera

manufactures a new and previously ignored market. A number of factors cause existing firms to ignore disruptive innovations and favor sustaining technology. Disruptive innovations tend to have low performance along traditional performance dimensions. Disruptive innovations also have lower profit margins and faster improvement rates, which gives them a catch-up advantage. In contrast, sustaining innovations tend to support existing firms with their high cost structures and well-defined product evaluation criteria.

Figure 3: HP Kittyhawk 1.3" micro drive and flash card

The Product Lifecycle

Products in a market tend to progress through a *product lifecycle* over time [39]. A product lifecycle is a series of sequential competitive stages that describe how products evolve over time and the customer's relationship with all of the comparable products in the market. The stages are functionality, reliability, convenience, and price (Figure 4 pg. 6). Each stage in the lifecycle identifies the primary attribute customers use to evaluate products. Customers judge products in the functionality phase superior if they have more desirable features. When most products in a market have a core set of desired features, customers' product evaluation criterion shifts to the next stage, reliability, in which higher-reliability products are judged superior, which drives buying decisions. When products are sufficiently functional and reliable, the customer's attention shifts to convenience. For technology products, this is often called ease-of-use. When all three of these criteria are met, products enter the commodity phase and compete primarily on price. An example is the commodity x86 processor market, in which processor cost is the primary concern for customers (assuming functionality requirements have been met such as performance, power consumption, etc.).

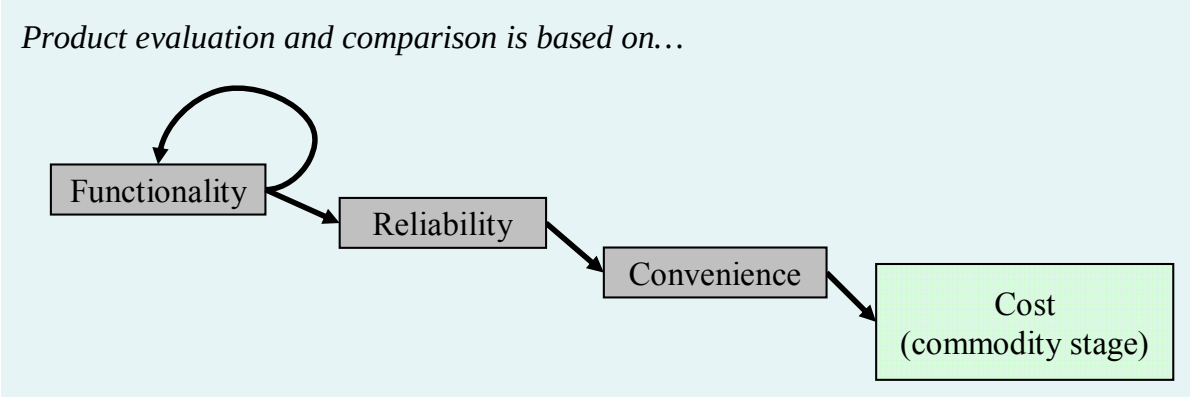

Figure 4: Product lifecycle

Changes in Product Lifecycle

The appearance of disruptive technology in a market often indicates a change in product lifecycle. In a competitive market, firms continually strive to increase the attractiveness of their products by improving them. Customers are willing to pay a premium for improved products. Firms can typically improve their products using sustaining technology faster than consumers demand these improvements. This is called performance oversupply [39]. Figure 5 shows lifecycle changes in the disk drive market over time. In Phase 1, competition was based on drive capacity. As sustaining technology improved drive capacity beyond demand, a disruptive technology—physically smaller drives—was introduced. This changed the basis of competition, and so it continues through to the commoditization phase.

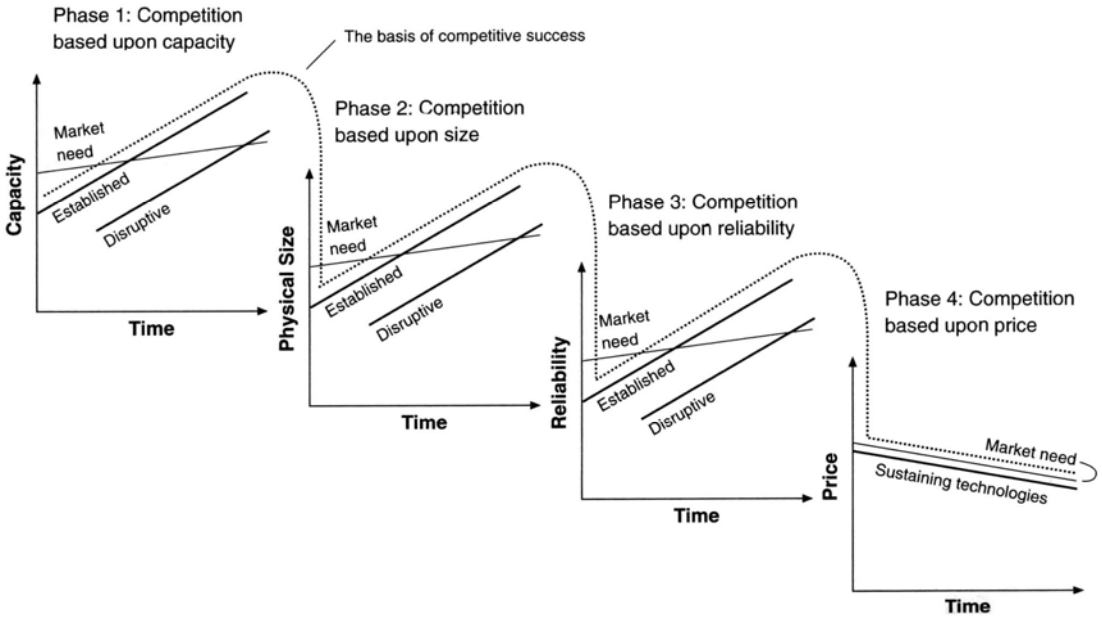

Figure 5: Lifecycle changes in the disk drive industry over time [39]

Firms strive to improve existing products with sustaining innovations because this protects their long-term product investments, and more importantly, it preserves their profit margins. It is difficult for an existing firm with high profit margins to enter a disruptive market with initial low customer demand and low profit margins, as they are simultaneously developing products that threaten their existing product lines.

Commoditization of the Computer Hardware Market

A relevant example of disruptive technology is x86 commodity hardware for the established RISC/Unix vendors. Commodity x86 hardware combined with Linux offers comparable performance for a fractionally lower price. A typical $25,000 RISC/Unix workstation can be replaced with a $3,000 x86/Linux workstation (see the FAA case study, pg. 44). Of the previously numerous RISC/Unix vendors, only three remain: Sun, IBM, and Hewlett-Packard. Sun is a classic example of an established vendor confronted with disruptive technology, shrinking demand, lower profit margins, and a high cost structure. A number of years ago Sun killed a plan to embrace x86 hardware. Sun is presently selling x86 hardware and offers an x86 version of Solaris, but one wonders if it may be too late—Sun continues to lose market share, is fighting larger competitors in a shrinking market, and is statistically unlikely to make a transition to a lower cost structure. Revenue has decreased 39% between 2001 and 2005 and 13,000 jobs have been cut. This is a long-term trend. Figure 7 (pg. 8) shows a MetaGroup study of platform growth through 2012 in terms of computing capacity. Unix is estimated to be 20% of the market in 2012 (13x growth), Linux 26% (251x), and Windows 51% (74x).

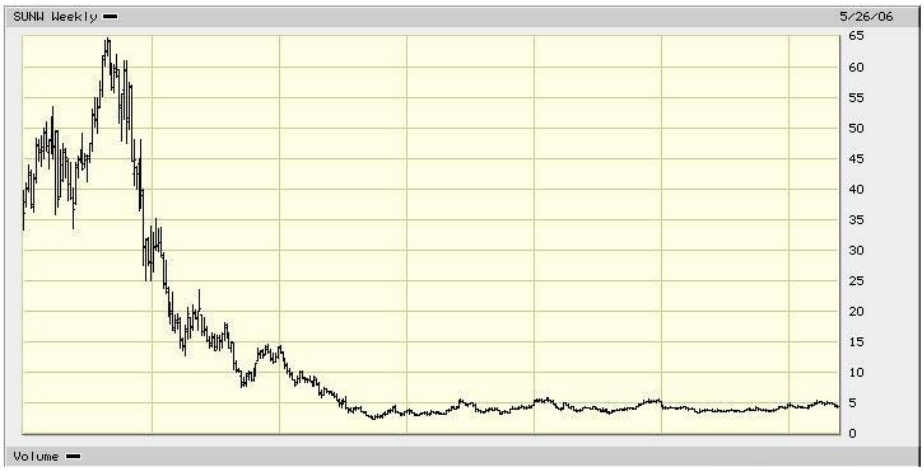

Figure 6: Sun Microsystems stock price, January 2000 to June 2006

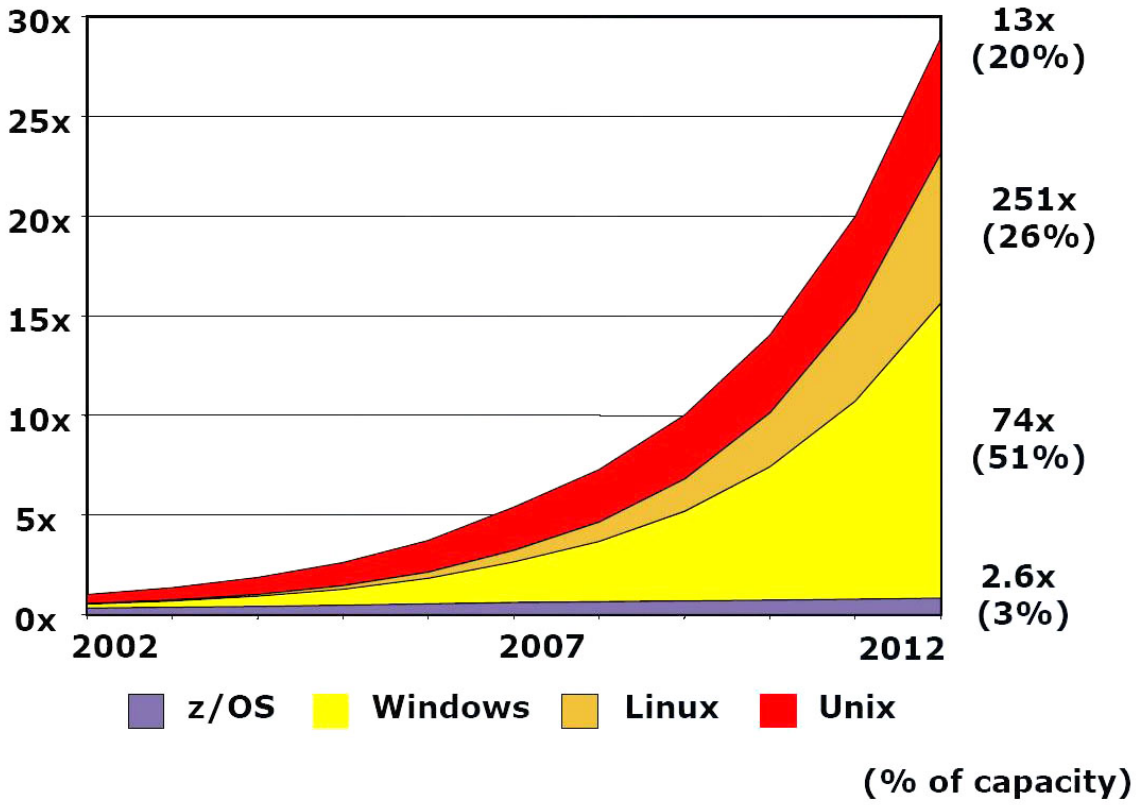

Figure 7: Data Center capacity growth 2002-12 [76]

SECTION 3:
HISTORY OF OPEN-SOURCE SOFTWARE

One of the goals of this report is to summarize the history, developments, strengths, and weaknesses of open-source software (OSS). To that end, this section starts with a few comments on OSS growth, which has been consistent. This leads naturally to two questions: what exactly is open-source software, and how did the OSS development model evolve? The answers to these questions are followed with some recent developments and usage statistics.

Open-Source Software Growth

The open-source software industry has a rich history full of interesting characters and diverse business interests, and is a global community connected by the Internet.[21] OSS growth has been consistent, organic, and strong. Firms such as Google, E*TRADE, Sabre, Travelocity, Yahoo!, and Amazon run their businesses with OSS and commodity hardware. The open-source Apache web server runs 67% of active web site servers [80]. A recent survey of Oracle users at the beginning of 2006 showed that 44% plan to be running their Oracle databases on the OSS operating system Linux by the end of the year [106]. Clearly, OSS is fulfilling a market need.

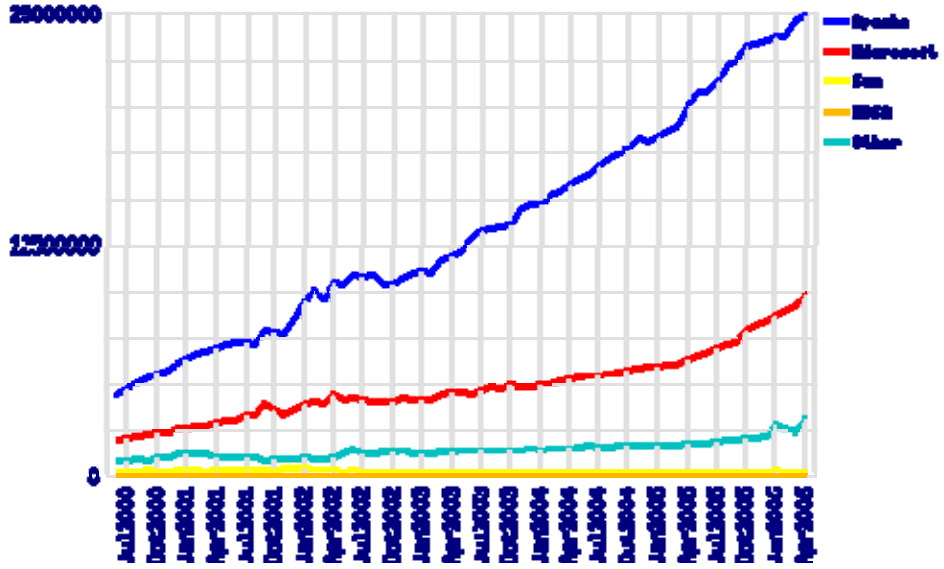

Figure 8: Active web servers, 6/2000 to 4/2006 [80]

Definition of Open-Source Software

What is open-source software? In essence, OSS is software that is distributed with its source code and is free. Open-source software is enhanced and maintained by a community, which is an organized group of unpaid volunteers who contribute, maintain, and enhance an open-source software project. The terms "open-source" and "free software" convey similar ideas but have historically slightly different origins. The term open-source is associated with the Open Source

[21] For detailed accounts of the origins of OSS, see E.S. Raymond, *The Cathedral and the Bazaar* (available at www.catb.org/~esr/writings/cathedral-bazaar/), and G. Moody, *Rebel Code*.

Initiative (OSI), a nonprofit organization founded by Eric Raymond and Bruce Perens in 1998 to promote open-source software [81]. The term was created with the hope that it would convey a more pragmatic business-friendly attitude than "free software." OSI defines criteria that software licenses must comply with to be considered "open source." Without getting bogged down in legal definitions, this generally means an open-source license must grant users the right to freely copy, use, and modify the software. It also stipulates that derivate works must be licensed under the same terms as the original.[22] Appendix D (pg. 89) discusses specific OSS licenses in more detail.

Aside from a legal definition, OSS has an implicit meaning to most people that assumes:

1. Anyone can contribute to the project and has full access to the software code.
2. A community of users and developers contributes to the evolution, development, and testing of the software.
3. There are rules about how the community operates (see Section 4).

Underlying these assumptions is the key idea that software code is a kind of knowledge that is freely shared with any interested person [107]. Collectively these ideas are sometimes referred to as the "open-source software movement."

History of Open-Source Software

The history of open-source software is an incremental story that has its roots in the creation of the ARPANET (the first packet-switched network), AT&T's Unix, Berkeley's BSD Unix, the GNU project, and the Linux open-source operating system. Netscape's decision in 1998 to open the source of their Internet browser was also an important event for the open-source approach. The development of open-source software is also a story of many interrelated events linked by the Internet.

In 1968, the Department of Defense Advanced Research Projects Agency issued an RFQ (Request for Quotation) for ARPANET, the first packet-switching network, and what would eventually come to be known as the Internet. Initially ARPANET connected primarily university computers and researchers, enabling the sharing of ideas and standards. For example, the Request For Comments (RFC) technical specification process was started in 1969 [61]. The next step in the process was AT&T's development of Unix in the early 1970s. From the beginning, AT&T distributed Unix source code and provided no support or maintenance, fostering an attitude of cooperation and mutual support among users [92]. Building on this tradition, in 1978 Berkeley started releasing nearly-free copies of their improved version of Unix, BSD Unix. Improvements from users were included in the system and made available in subsequent releases, which, combined with the low price, made BSD very popular [91]. AT&T and the University of California Berkeley subsequently became entangled in a lawsuit, over copyright issues, that was not resolved until much later. The uncertain legal status of Unix provided critical impetus for two software projects crucial to the future of open-source: GNU and Linux.

[22] For OSI's Open Source Definition, see www.opensource.org/docs/definition.php.

In 1971 Richard Stallman, a programmer, was working in MIT's artificial intelligence laboratory, which had a rich tradition of sharing source code and ideas among developers. Because of corporate involvement with the lab and subsequent intellectual property disagreements, Stallman came to believe that there should be no limitations on access to software code and argued against any type of intellectual property. In 1983, Stallman announced the GNU (GNU's Not Unix) project, which has a goal of creating an entirely free and open-source clone of the Unix system. Stallman's critical innovation was the GNU General Public License (GPL), which was released in 1989. The GPL is a copyright license with special qualities intended to protect a user's right to distribute and modify a work. The GPL also protects communal rights to derivative works—derivative works are by definition copyrighted with the GPL. Open-source copyright licenses with this property are often called *copyleft* licenses.[23] Most copyright licenses restrict users from doing certain things; for example making copies of software and giving it to friends. The GPL is different in that it specifies rights that Stallman wants copyright licensees to have. For example "You may copy and distribute verbatim copies of the Program's source code as you receive it..." [52]. GNU software and many OSS projects license their software using the GNU GPL, including the open-source operating system Linux.

Linus Torvalds was a Finnish computer science student in Helsinki in 1991 when he placed on the Internet the source code for version 0.01 of a free operating system he was developing as a hobby. It was called Linux. Torvalds was using the GNU compiler and other GNU tools, and was inspired in part by Minix, an operating system developed by Andrew Tanenbaum for teaching computer science students about operating systems. From the beginning, Torvalds solicited feedback from people connected by the Internet. The Internet and Torvalds' willingness to incorporate other people's suggestions and improvements encouraged people to contribute [77]. Discussing early Linux bug fixes, Torvalds remarks that "they started out so small, that I never got the feeling that, hey, how dare they impose on my system. Instead, I just said, OK, that's right, obviously correct, so in [to Linux] it went. The next time it was much easier, because at that time there weren't many people who did this, so I got to know the people who sent in changes. And again they grew gradually, and so at no point I felt, hey, I'm losing control" [77]. In this way, Linux became two things: an operating system, and a new distributed, collaborative, and parallel development model (see Section 4 for details). Before the success of Linux, no one could have imagined that it was possible to develop something as complex as an operating system using volunteers connected mainly by the Internet. As early as 2001 it was estimated that more than 40,000 developers worldwide had contributed to Linux [89].

Another important point in the history of OSS is January 1998, when Netscape Communications turned their Navigator Internet browser into an open-source product. At the end of a long market-share losing battle with Microsoft, Netscape had concluded that the only way to survive was to play a different game. For many, Netscape's decision legitimized the open-source approach as a strategy for competing with a proprietary software vendor. One year later Hewlett-Packard, IBM, Dell, Intel, and SGI announced support and official approval of Linux running on their hardware. The corporate embrace of OSS was underway.

[23] For a detailed explanation of copyleft, see www.gnu.org/copyleft.

Table 2: Motivation for Linux server adoption [7]

Linux Motivators for Server Usage	%
Relatively low cost or no licensing fee	78
Reliability	74
Performance	73
Windows security issues	65
Need an alternative to Windows	60
Recommendations by our technical staff	60
Ability to modify source code to meet our needs	45
Development tools widely available through the Internet	45
Fulfills company requirements or standards	40
Fast software patches and bug fixes	40
Measurable Return on Investment (ROI)	38
Company has an open-source philosophy	33
Unix	25

Recent Developments

The current dynamics in the Linux market provide insight into the larger OSS market. The primary driver of Linux adoption on both desktops and servers is low cost, according to a survey in early 2005 [7]. This is significant because it indicates OSS has reached approximate feature and convenience parity with proprietary products (see Section 2). The top three reasons cited for Linux use on servers are low cost (78%), reliability (74%), and performance (73%). These growth drivers indicate a mature product. Other reasons are shown in Table 2. The OSS market is healthy with many competitors in different product categories and continues to respond to market demands. A partial list of popular open-source projects is shown in Table 3 (pg 13). In 2002, support was the primary concern, but this is no longer viewed as an issue due to support programs implemented by many vendors. On the desktop, where Linux has 9% market share, key desired features are installation convenience and device support. A relatively new Linux distribution named Ubuntu (oo-BOON-too) with the motto "it just works" is focusing on the desktop market and is receiving positive attention.[24]

Linux has many areas to improve, as Table 7 (Section 5) shows. The top three reported problems with Linux on servers are technical knowledge of staff (35%), compatibility with existing software (33%), and problems related to multiple versions of Linux (24%). Approximately one-third of respondents indicated no problems were encountered moving to Linux (28%).

[24] See www.ubuntu.com.

Table 3: Partial list of open-source projects from Appendix B

Open-source Project	Category	License	Reference
Apache	Web Server	Apache	www.apache.org
BSD variants	Operating System	BSD	en.wikipedia.org/wiki/BSD
Cygwin	Unix-like Environment	Modified GPL	cygwin.com
Drools/JBoss Rules	Rules Engine	Apache	labs.jboss.com/portal/jbossrules
Eclipse	Development Platform	Eclipse	www.eclipse.org
EnterpriseDB	Database	GPL	www.enterprisedb.com
GCC	Compiler	GPL	gcc.gnu.org
JBoss Application Server	J2EE Server	LGPL	www.jboss.org
Jena	Semantic Web Toolkit	Jena	jena.sourceforge.net
Linux	Operating System	GPL	en.wikipedia.org/wiki/Linux
MapServer	Internet Map Server	MapServer	mapserver.gis.umn.edu
MediaWiki	Collaboration Application	GPL	www mediawiki.org/wiki/MediaWiki
Mono	.NET Application Framework	GPL, LGPL	www.mono-project.com
Firefox	Web Browser	Mozilla	www.mozilla.com
Thunderbird	Email Application	Mozilla	www.mozilla.com
MySQL	Database	GPL	www.mysql.com
OpenSSH	Communications Tool	OpenSSH	www.openssh.com
OpenSSL	Communications Tool	OpenSSL	www.openssl.org
Open Office	Office Productivity Suite	LGPL	www.openoffice.org
Perl	Development Language	GPL	www.perl.org
Python	Development Language	Python	www.python.org
PHP	Development Language	PHP	www.php net
PostGIS	Database Extension	GPL	www.refractions net
PostgreSQL	Database	BSD	www.postgresql.org
Protégé	Ontology Editor	Open Content License	protege.stanford.edu
rdesktop	Remote Desktop Client	GPL	www rdesktop.org
Samba	File and Print Sharing	GPL	www.samba.org
VNC	Remote Desktop Client and Server	GPL	en.wikipedia.org/wiki/VNC

The use of OSS is growing, and large traditional IT vendors such as IBM, Hewlett-Packard, and Oracle view OSS and Linux as strategic. Among software developers, the OSS database MySQL is approaching majority market share (44%), which is an increase of 25% between April and October of 2005 [40]. The Independent Oracle Users Group (IOUG), in a survey in early 2006, found that 44% of respondents will be running their Oracle databases on Linux in the next twelve months [106]. The president of IOUG said "now the market has accepted Linux. [It] has accepted [that] you can run hugely scalable, Oracle RAC (Real Application Clusters), multi-node, thousands-of-concurrent-user Oracle instances, on Linux...[combine that with] the overall lower cost of ownership, and you've got the best of both worlds, and we're seeing the results" [106]. In addition, Linux is currently in use as a server operating system at 49% of companies polled [7]. The remainder are either pilot testing or will be in twelve months (23%) or have no plans (28%). Both IBM and Oracle certify and support their databases on Linux. Oracle

sells Linux versions of all Oracle products. IBM runs and supports Linux on its mainframes, POWER servers, workstations, and PCs. IBM presently has 300+ full-time developers contributing to Linux projects and has a large Linux budget [53,90].

The OSS approach continues to drive industry change. Microsoft is running a 300-server OSS lab to test interoperability with OSS products and regards Linux as one of its biggest threats [62,67]. Microsoft has also created a web site (CodePlex[25]) that is a community development repository for Microsoft .Net ("dot-net") developers. To compete with the OSS database MySQL, Oracle, IBM, and Microsoft have lowered their database prices and offer restricted free versions to attract customers [66]. Oracle has chosen to "bear hug" OSS rather than fight it, with Oracle's CEO, Larry Ellison saying "we are moving aggressively into open source. We are embracing it. We are not going to fight this trend. We think if we're clever, we can make it work to our advantage" [65]. Oracle has recently purchased OSS companies Sleepycat Software and Innobase, with the strategy of making revenue through service and support rather than initial or ongoing licensing fees. Sun has recently announced its intentions to open-source the Java development platform. This is seen as a response in part to Linux vendor Red Hat's recent purchase of the popular Java middleware provider JBoss.

A striking example of the OSS approach driving industry change is EnterpriseDB[26], an open-source relational database system that is compatible with Oracle. EnterpriseDB is based on the OSS PostgreSQL database and consists of a database server, replication server, migration tool set, developer studio, debugger, and management server. Sony Online has implemented EnterpriseDB and converted more than 150 existing Oracle 9i databases used for online gaming [42]. EnterpriseDB was started in 2004 and is based in New Jersey. They have received significant venture capital funding. EnterpriseDB uses a dual-license scheme and the GNU GPL license[27].

OSS use within government is expanding. A number of cities worldwide have standardized on OSS, including: Munich, Germany; Turku, Finland; Rome, Italy; and Mannheim, Germany. In Europe, a Maastricht University survey of twelve countries found 49% of European local government bodies are using OSS and 70% report an expectation of further increase [41]. Almost one-third (29%) did not know their local government was using OSS. In the U.S. there is movement among State Chief Information Officers (CIOs) for the definition of open standards relating to document formats [23]. In early 2006 Minnesota introduced a bill requiring the use of open data formats for archiving State documents [68]. This follows Massachusetts' adoption of the Open Document Format (ODF), which is a standardized office productivity data format approved by the OASIS consortium (Organization for the Advancement of Structured Information Standards). The ISO (International Organization for Standards) and the IEC (International Engineering Consortium) have also approved ODF. The close association between the standardization of data formats and use of OSS is not coincidental. Both have shared goals, as explored in the next two sections.

[25] See www.codeplex.com.
[26] For further information, see www.enterprisedb.com.
[27] Appendix D describes dual copyright licensing (pg. 89).

SECTION 4:
HOW OPEN-SOURCE SOFTWARE IS DIFFERENT

Open-source software and its development process are fundamentally different from the closed-source approach, and are often initially difficult to understand. This section explores how OSS is different, starting with a few brief perspective-enhancing comments about software engineering and the low success rate (29%) for software projects in general. Understanding software development's low success rate is important for understanding some of the advantages and disadvantages of the OSS development model discussed in this and subsequent sections. A key distinction is made between software that is differentiating or non-differentiating—one is beneficial to share with competitors, the other is not. This is followed with a comparison of the strengths and weaknesses of the OSS and closed-source development models. This is important for understanding how the OSS model tends to reduce software lock-in. This is followed by a more detailed explanation of the OSS development process and innovation.

The distinctiveness of OSS begins with three assumptions:

1. Anyone can contribute to the project and has full access to the software code.
2. A community of users and developers contribute to the evolution, development, and testing of the software.
3. There are rules about how the community operates.

These assumptions naturally lead to a number of questions. If the source code is open, where is the value? What keeps absolute chaos from breaking out? What stops someone from taking the source code and starting a new project? How is quality assured? If there is no profit motive, what inspires innovation? This section will answer these and other questions. First, some comments about software engineering are provided as background.

Software Engineering

We take it for granted that approximately one-third of software projects are successful, an abysmally-low success rate. If any other engineering profession had similar failure rates (e.g. construction of buildings, bridges, medical devices, cars, airplanes, etc.) a National emergency would be declared. Clearly, software development is high-risk. This is relevant and important for ATMS projects because any methodology or approach that lowers development risks is valuable. The low software project success rate is due in part to the young age of the profession. Software Engineering as a profession is only about 30-40 years old, and is still emerging from its craftsman/hacker stage into a full engineering discipline. Other engineering professions have successfully made similar transitions. *Software Engineering* is the use and development of sound engineering principles for the production of software that is reasonably priced, reliable, and works efficiently on real machines [88]. The Standish Group surveyed more than 50,000 completed software projects in the U.S. between 1994 and 2004 [4]. For the year 2004 it found 29% of projects succeeded (within budget, with required features, on time), 53% of projects were challenged (completed late, over budget, and/or with less than desired functionality), and the remaining 18% failed (were canceled prior to completion). In 2000, cost overruns averaged 45%, time overruns were 63%, and delivered features were 67% of those planned. The dollar costs

associated with this low success rate are huge—estimated at $81 billion spent on canceled projects alone in 1995.

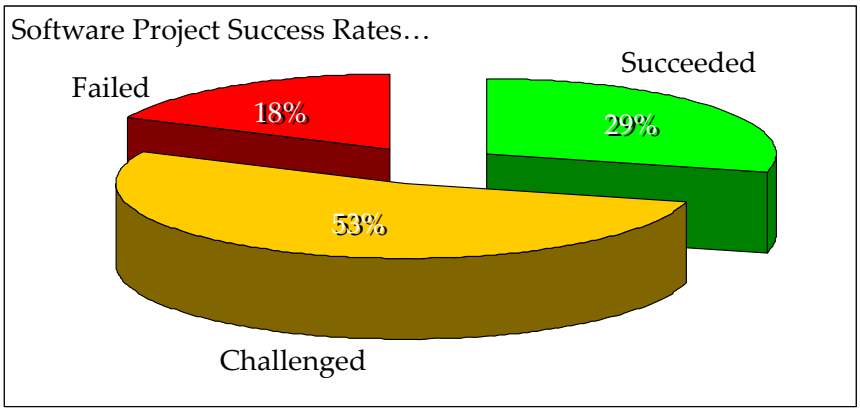

Figure 9: Software project success rates in 2004 [4]

Differentiating and Non-Differentiating Software

If open-source is a good thing, should an organization share all of its software code? The answer is no—it depends on the software, and whether it is *differentiating* or *non-differentiating* software. Differentiating software distinguishes a company from its competitors in the minds of potential customers [87].[28] For example, AutoCAD is differentiating software for Autodesk's customers. It is generally not a good idea to share differentiating software with competitors. On the other hand, it is often beneficial to share non-differentiating software with competitors. For example, all legal firms require nearly the same functionality from accounting software. Enlightened legal firms might agree to cooperate and share the development of accounting software and simultaneously continue to compete in the domain of legal services. New functionality added by a single person or firm would be available to every other firm at no cost. In summary, sharing non-differentiating software increases efficiency by sharing costs and development risks. This enables additional resources to be spent on differentiating software if desired, which enhances an organization's appeal to customers.

Types of Software Development

Different types of software development have strengths and weaknesses and it is useful to distinguish between them. Perens defines four types of software development and benefits and weaknesses of each approach [87]. These are:

- Retail
- In-House

[28] This distinction is also useful for knowledge and manufactured items. Economics consider both software and manufactured items (e.g. hammers) to be embodied knowledge (see Välimäki). A hammer is both the knowledge required to build it and the physical object. In a similar way, software is both the knowledge required to develop it and a program running on a physical machine.

- Collaboration (closed to the outside world)
- Open-Source (open to the outside world)

Retail and open-source development will be discussed briefly. Retail software development is often called proprietary because source code is almost never available and the vendor maintains complete licensing control over the source code. Familiar examples are products from Apple, Microsoft, Oracle, etc. Retail software development is only approximately 25% of the total software market by dollar value [1]. Retail software developers assume complete responsibility for the costs and risks of developing new products. These costs and risks are high, which tends to push retail software vendors towards high-volume markets, for example, word processors, games, etc. Finally, retail developers spend a small amount of their total revenue on research and development. For example, Oracle spent 12%, 13%, and 13% of total revenue in 2003, 2004, and 2005, respectively, on research and development. That means for every dollar spent on retail software, only thirteen cents went into the research and development of the purchased product. The remaining 87 cents went into advertising, sales, profits, etc.

In contrast to retail development, OSS tends to distribute development risks among contributors. If a new feature is desired in an existing product, it is likely that multiple contributors will be interested in participating. This distributes development risks and costs among participants. For new projects with no collaborators, development risks would be similar to the retail model. For this reason, OSS projects tend to be incremental and avoid starting from scratch. In this regard, the large number of OSS projects available creates a kind of competitive market of available source code for starting new projects. Finally, OSS is more efficient compared to retail development—every dollar spent on software is a dollar spent on software development. See Table 3 (pg. 17) for a comparison of development methods.

Table 4: Retail versus open-source software development models [87]

Attribute	Retail	OSS
Distributes development risks	No	Yes
Distributes development costs	No	Yes
Overhead rate	High	Low

Software and Lock-in

Lock-in is a term used in economics to describe a situation in which a customer or vendor faces high costs to switch products or technologies. For example, there is a high re-training cost to switch word processors or operating systems, or to learn to drive safely on the opposite side of the road. Traditional vendors face lock-in when confronted by competitors using new disruptive technology—their high cost structure makes switching costs prohibitively expensive. There is little lock-in associated with, for example, toothpaste or calculus textbooks. In economic terms software lock-in occurs because the price customers are willing to pay for a product they have spent time learning (sometimes years) becomes proportionally more inelastic (see Appendix A). This customer dependency encourages vendors to raise prices to the maximum bearable level. By offering a free or low-cost substitute, an OSS product will tend to have a damping influence on proprietary product cost.

One of the primary benefits of OSS is greatly reduced lock-in. For customers, software lock-in has two painful aspects: 1) the ongoing cost of a software license the customer is locked into, and 2) the learning and development cost of switching to a new product. OSS eliminates the first concern because there is no acquisition cost for OSS. That leaves the second concern, the development and learning costs associated with switching products. OSS reduces the pain associated with this concern in a number of ways, primarily because OSS projects, developers, and users are similarly motivated. First, there is no motivation to introduce nonstandard hooks to increase lock-in, because there is no revenue motivation. Second, OSS projects, developers, and users are inherently more interested in following and forming standards because this increases the desirability of the software to others, which encourages further contributions. The standards-following nature of OSS is an especially strong incentive for State, Federal, and local governments, which are often charged with following standards. Third, there are typically a number of competing products in any given OSS market. For example in the OSS database market, there are MySQL, PostgreSQL, Ingres, Firebird, and MaxDB[29], among others. Fourth, OSS projects tend to be portable because their source code is available. For example, moving a project from Linux to BSD to proprietary Unix to a POSIX environment on Windows XP is possible. Many OSS projects are developed for all of these platforms simultaneously from a single code base. For example, switching a MySQL database from Windows XP to Linux is relatively easy.

The Open-Source Development Process

When Torvalds started Linux in 1991, other popular operating systems already existed, such as Minix, with available source code. Linux is the one that grew into a worldwide phenomenon running on everything from traffic controllers to supercomputers, and the one embraced by the corporate world. Torvalds' key innovation was a new development process—a new combination of relaxed top-down guidance mixed with developers and users connected over the Internet contributing numerous self-motivated suggestions, requests, fixes, and new features. This approach takes maximum advantage of a distributed development community. The following paragraphs briefly discuss OSS project organization, projects traits, and the OSS innovation process.

OSS project organization varies by project. Contributors regard themselves as part of a community [47]. Typically, each project has a single recognized leader responsible for the overall direction and vision of the project, although there have been other structures. Leaders are responsible for finding competent replacements if they no longer wish to lead. Leaders may or may not write actual software code. An OSS copyright license (e.g. GNU's GPL) is typically used, ensuring distributed ownership of source code. This means leaders do not own anything and must therefore lead without coercion. Ultimately all a leader has is the respect and admiration of the community combined with his exclusive right to determine the contents of subsequent versions of the project [89]. Assisting a leader are developers who write code, make suggestions, and find and repair defects. On large projects, developers may be organized hierarchically. Developers may be volunteers working for free, or employees contributing as part of their job function at a corporation or government agency. Members of the community also include users who may or may not report defects and provide suggestions and requests.

[29] See www mysql.com, www.postgresql.org, www.ingres.com, www.firebirdsql.org, and www.sapdb.org.

Dissatisfied community members may at any time start a new parallel project with a new name, using any version of the existing project. This is called *forking* a project. This is a rare occurrence and typically only happens in extreme situations where a large number of community members are highly dissatisfied. The potential of a project forking is positive for users and developers because it encourages alignment of developer and user interests [78]. New projects live and die based on the number of contributors they can attract.

Eric Raymond has famously described some of the traits of successful OSS development [89]. The majority of these guidelines are focused on enhancing the distribution of project work—in other words, enhancing a bottom-up approach. The guidelines also help create a giant positive feedback network, which ultimately encourages participation. These development guidelines include:

- Consider code contributors to be co-developers. This encourages further contributions and a sense of collective ownership. This also indicates the importance of listening to suggestions and requests from users and developers. Contributions are accepted from anyone, lowering barriers to community entry.
- Release new versions of the project often. This encourages a sense of improvement, responsiveness, and relevance.
- Consider testers to be your most valuable resource. The more testers and users a project has the more defects are found, and the more everyone benefits.
- Focus on recognizing good ideas from the community, rather than your own good ideas.

Innovation and Open-Source Software

Innovation is often cited as another OSS strength. Joode describes two opposite approaches for generating innovation: the *rational* process, and the *variation and selection* process [61]. The rational approach is top-down and involves analysis and consideration of available options. The second approach is used by OSS, and tends to be a bottom-up approach to creating innovation. It relies on the creation of variation combined with selection among alternatives. The OSS development process creates variation through 1) the availability of source code, 2) low barriers for personal participation, 3) a large and diverse group of contributors, and 4) a pervasive pragmatic attitude among group members. These factors and a lack of restrictions create a high degree of variation. Selection among this variation is a result of the cumulative effect of individual choice and is based on *professional attention*. In the OSS world, professional attention cannot be coerced. Many factors affect the personal interest of OSS project contributors: the popularity of a project (number of downloads, number of contributors), the reputation of those involved, the existence of competitors, personal interest, the ability of the project leader to distribute success and recognition, and so on. New useful features and enhancements generate interest and attention, resulting in incorporation into subsequent versions. Because source code is available, innovations can more easily flow between different OSS projects. The innovative nature of the OSS development process is particularly suited for collaboration between government and research organizations.

Typical Open-Source Development Environment

This subsection briefly compares a proprietary software development environment with an OSS development environment. The focus is on matching proprietary products with OSS products within the product category. In some product categories there are many OSS products available, which can make counterpart selection complex. For example there are at least eighteen OSS Java virtual machines (JVM), some specialized.[30] The goal here is to suggest a mainstream OSS counterpart for each proprietary product, not provide an exhaustive list of candidates.

Table 5 shows COTS and potential OSS product counterparts. Some of these OSS products are described briefly in Appendix B (pg. 79). Most of the listed products are cross-platform, raising the possibility of a heterogeneous environment. For example, EnterpriseDB supports Linux x86, Linux x86-64, Mac OS X Intel, Mac OS X PPC, and Solaris 10 x86-64. In general, OSS counterparts should not be viewed as drop-in replacements for COTS products. OSS products that emphasize compatibility are the Java virtual machines and class libraries, the Oracle compatible database EnterpriseDB (pg. 80), Mono (pg. 82), and the language compilers.

Some COTS products from Table 5 are rather specialized, such as the Gensym G2 system and SL-GMS widget framework. Typical Linux and BSD distributions are bundled with a large number of utilities that Solaris or HP-UX users would be familiar with: grep, awk, sed, ping, man, more, cat, vi, Emacs, finger, NFS, etc. Shells include Bash (Bourne), PDKSH, TCSH, Z, and more. Several GUIs are typically included in standard distributions such as GNOME and KDE. The Eclipse development environment supports many languages such as Java, C/C++, Fortran, PHP, Perl, Ruby, COBOL, UML2, and Python, among others.

In operational terms, a machine running Linux or BSD will fit into an existing computing environment, necessitating file and printer sharing. Typical Linux and BSD distributions are bundled with Samba, an OSS file and printer sharing application. Samba enables Linux machines to appear as Windows file and print servers on a network. Samba also enables simple file sharing between Linux and BSD machines. NFS is also supported. Rdesktop is an OSS remote desktop client that allows remote access to Windows machines running a remote desktop server (see pg. 85). VNC products are also available that enable remote connections between many types of machines (see pg. 85).

[30] For a list of OSS Java virtual machines, see en.wikipedia.org/wiki/List_of_Java_virtual_machines.

Table 5: COTS and potential OSS product counterparts

Company	COTS Product	Category	Potential OSS Counterparts
Gensym	G2 7.0	Object oriented rule based system	JBoss Rules (Drools)
Gensym	G2-Oracle Bridge	G2 database connectivity	PostgreSQL ODBC, JDBC MySQL ODBC, JDBC
Gensym	GSI	C/C++ external system interface	For Java: JNI
HP, Sun	HP-UX or Solaris	Operating System	Linux, BSD
Microsoft	.NET	Development Framework	Mono
Oracle	Oracle Database 9.2	Relational Database	MySQL, PostgreSQL, EnterpriseDB
Oracle	Oracle 9i Application Server	J2EE, Web services, JSP Web server, JVM, HTTP Server	See Java EE below, Apache Axis, Apache Tomcat, See Java SE below Apache HTTP Server
Oracle	PL/SQL	Oracle procedural language extensions to SQL	EnterpriseDB, PostgreSQL: PL/pgSQL
Oracle	Pro*C	C/C++ preprocessor for SQL	For PostgreSQL: ecpg, For MySQL: C interface
Oracle	JDeveloper	Integrated Java development environment	Eclipse
SL	SL-GMS	Widget Development Framework	For Java: SWT, Swing For C++: Qt
Sun	Java EE	JRE Enterprise Edition, J2EE	JBoss AS, Apache Struts, Spring Framework, Sun GlassFish
Sun	Java SE	JRE Standard Edition, J2SE	Sun Java SE[31] Apache Harmony, GNU GCJ
Tibco	SmartSockets 6.60	Reliable messaging	Apache ActiveMQ
Various	C/C++	C language compiler	GCC

[31] Sun Java SE is not an open-source product but is free. See java.sun.com/javase/faqs.jsp#Licensing.

SECTION 5:
OPEN-SOURCE SOFTWARE STRENGTHS

This section discusses some of the strengths of the open-source development model: reduced lock-in, reliability, security, development efficiency, how OSS might positively affect ITS markets (more specific to the ATMS application focus), and the use of standards in OSS. This is a subset of benefits cited in other sources. For example, in a report for the Department of Defense, Mitre Corp. identified eight key strengths (Table 4) [63]. In a survey of CIOs the three greatest strengths of OSS were reported to be lower capital investment (63%), lower total cost of ownership (59%) and greater reliability (41%) [113]. For government, perhaps one of the most important OSS benefits is reduced lock-in. An important point is that OSS has many of the traits of disruptive products (see pg. 4) —it is less expensive, more reliable, and improving at a faster rate than traditional products.

Table 6: OSS strengths for DoD, reported by Mitre Inc. [63]

OSS Strengths for Department of Defense
Massive programming expertise
Research and development covered by volunteer labor
Accepted leadership structure
Quick release rate
Parallel development and debugging
Maturity of code
Culture of sharing
Long-term accessibility

Reduced Lock-In

Lock-in is also discussed in detail in Section 4. Briefly, lock-in is a term used in economics to describe a situation in which a customer or vendor faces high learning or financial costs to switch products or technologies. OSS greatly reduces lock-in for four reasons: 1) there is no financial motivation for OSS developers to increase lock-in; 2) OSS projects and users tend to follow and form standards; 3) the OSS market is healthy and competitive with a number of products in any given market segment; and 4) OSS projects tend to be portable because their source code is available.

Reliability

It has been estimated that software defects cost the U.S. economy $59 billion annually [2]. The OSS development process creates software that 1) tends to have fewer defects and 2) repairs discovered defects faster. The basis of the OSS development process (Section 4) is peer review of code and the ability of anyone to find defects and forward patches to the developers. This creates a rich feedback loop. This has been famously stated by Raymond as "given enough eyeballs, all bugs are shallow" [89]. Peer review is crucial in all engineering disciplines. For software engineering, it has been estimated that peer review catches approximately 60% of defects [29]. Given the abysmally-low success rate of software projects (29%) and the high level of defects, a development methodology that is inherently based on peer review with the widest

possible audience is significant. In fact, any software development practice that moves current software development practices from the craftsman stage towards a full engineering discipline is significant.

The defect reducing nature of a peer-review-based development process can be quantified. The Department of Homeland Security is funding a three-year $1.24 million study "Vulnerability Discovery and Remediation Open Source Hardening Project" [79]. This program is part of a larger Federal effort to perform security audits of approximately 40 open-source software packages such as Linux, MySQL, and Apache. The project uses automated source code scanning tools (from Coverity Inc.) which are based on research by Stanford University. Table 5 shows defect rates for popular OSS projects and typical commercial software. The mean number of defects per 1000 lines of source code in 32 open-source projects was found to be 0.434 [38]. MySQL was found to have .224 defects per 1000 lines of code, which is four times better than typical commercial software. It is worth noting that the OSS projects in Table 5 are the most popular in terms of the quantity of users. Clearly, there is an inverse relationship between the number of users and the defect rate. In addition, the OSS development process produces rapid repairs of discovered defects. "More than 900 flaws were repaired in the two weeks after Coverity, which makes tools to analyze source code, announced the results of its first scan of 32 open-source projects. As a result, some of the software is now entirely bug free" [50].

Table 7: Coverity Inc. defect rate study results [38]

Development Model	Project	Defects per 1000 Lines of Code
OSS	Perl	.186
OSS	MySQL	.224
OSS	Linux	.233
OSS	Apache	.250
OSS	Python	.372
OSS	PHP	.474
OSS	LAMP Average	.290
OSS	Baseline OSS	.434
COTS	Typical Commercial Software	.896

Security

When considering the security of any system, healthy skepticism is in order. Ultimately, security depends on knowledgeable, skilled, and paranoid system administrators. That said, superior security is widely associated with OSS. However, evaluation of security is difficult because of the large number of variables: severity of defects, expertise of system administrators and users, how rapidly vendors respond to reported and unreported problems, and the number of attacks targeting the platform in question. Higher volume systems will naturally experience more attacks of greater diversity. Both proprietary and OSS vendors have used the complexity of the issue to make competing claims of superior security.

A qualitative approach is one way of evaluating security. Using this approach indicates a clear security preference for OSS among system administrators and CIOs over proprietary

software [63]. However, there is healthy skepticism that disclosing source code makes the system more secure [109]. There is also healthy skepticism that not disclosing source code provides too great of a temptation for proprietary software vendors to hide numerous and critical security vulnerabilities.

A wide-ranging analysis report prepared for the Department of Defense (DoD) in 2003 regarding their use of *Free and Open Source Software* (FOSS) concluded [33]:

- The DoD should encourage the use of OSS. OSS "applications tend to be much lower in cost than their proprietary equivalents, yet they often provide high levels of functionality with good user acceptance. This makes them good candidates to provide product diversity in both the acquisition and architecture of DoD systems."
- The DoD should create a list of "Generally Recognized As Safe" OSS that is commercially supported, widely used, and has a demonstrated record of security and reliability.
- "Banning FOSS would have immediate, broad, and strongly negative impacts on the ability of many sensitive and security-focused DoD groups to defend against cyber attacks."
- Banning FOSS "would remove the demonstrated ability of FOSS applications to be updated rapidly in response to new types of cyber attack."
- "FOSS software plays a more critical role in the DoD than has generally been recognized...one unexpected result was the degree to which security depends on FOSS."

The speed with which discovered security vulnerabilities are repaired is important. Quantitatively, OSS security advisories are repaired faster. The corporate security provider Secunia Inc. tracks and reports security advisories for a number of OSS and proprietary software products [27]. Table 6 shows the number of unpatched and patched security advisories for a number of products. The data shows that the OSS products were more or less completely patched. At the time of this writing, the LAMP stack consisting of Red Hat ES 5/SUSE, Apache, MySQL, and PHP contained two unpatched advisories. Furthermore, Microsoft's Windows XP over a five-month period reduced its number of unpatched advisories by one, down to 28. Some of these advisories are rated as "highly critical."

Many of the traditional arguments that OSS is inherently more secure are based on the strengths of the OSS development process:

- The ability of anyone to find and repair problems because the source code is available. This results in the rapid repair of defects and security advisories [38].
- Peer-reviewed software code results in a lower number of defects. A lower number of defects means a lower number of potential security exploits.
- The availability of source code enables anyone to enhance security. For example, the National Security Agency (NSA) developed Security Enhanced Linux[32] (SELinux)

[32] See www nsa.gov/selinux.

which is now a part of some mainline Linux distributions. SELinux implements mandatory access control within the kernel.

Table 8: Secunia Security vulnerability report [27]

Development Model	Product	Date	Unpatched Security Advisories	Patched Security Advisories
COTS	Microsoft Windows XP Professional	01/2006	29	95
OSS	Red Hat 9 Linux	01/2006	1	99
OSS	SUSE Enterprise Server 9	01/2006	0	91
COTS	Microsoft Windows XP Professional	05/2006	28	106
OSS	Red Hat Enterprise Linux ES 4	05/2006	0	91
OSS	MySQL V5	05/2006	1	2
COTS	Oracle Database 10g	05/2006	4	11
OSS	Apache HTTP Server V2.2.x	05/2006	0	0
OSS	PHP V5.1.x	05/2006	1	6
COTS	Microsoft Internet Explorer V6.0.x	05/2006	21	101
OSS	Mozilla Firefox V1.x Internet browser	05/2006	3	30

Efficiency

OSS can potentially enhance developer efficiency, organizational efficiency, and macro-economic efficiency [87,104]. On a National level, one can imagine the benefits from a National ATMS open-source effort in which individual State DOTs contribute their best functionality or component. Potentially, each DOT could benefit from the best contribution from every other state. For large organizations, the easy accessibility of OSS via the Internet and a simple download enhances efficiency. No purchase requisition forms are required—this can be a significant time and cost savings in an organization. An individual with time, skill, and an Internet connection can solve previously unsolvable problems.

For the individual developer, it is important to make a distinction between using OSS products and working on a project that is itself an OSS project. Most efficiency gains for individual developers result from reuse of source code that is open. Efficiency gains are therefore proportional to the quantity of open-source code available. A more subtle and equally powerful efficiency enhancer is *specialization*, also known as division of labor.[33] Analogous to free trade between nations, specialization allows developers to contribute to a collective project in the way they deem most appropriate, enhancing organizational and National efficiency.[34] For an

[33] Division of labor is the "specialization of cooperative labor in specific, circumscribed tasks and roles, intended to increase efficiency of output." (see Wikipedia, 'Division of Labor'). Adam Smith argued in *An Inquiry into the Nature and Causes of the Wealth of Nations* (1776) that the primary benefit of trade between nations results from the division of labor. One can imagine similar benefits between organizations sharing code.

[34] An example of specialization is the ability of some people to rapidly identify defects and the ability of others to rapidly fix them. Another example is the development of the first port of Linux to Sun's SPARC. See G. Moody, *Rebel Code*.

organization, this enables developers to spend more time enhancing differentiating software (see Section 4), increasing the value of the organization to its customers.

Healthy ITS Markets

A healthy ITS market is one of the six primary goals of the National ITS program [6]. A market includes both buyers and sellers [98]. A healthy market results when benefits for both buyers and sellers are maximized over the long run. It is important to note that ultimately customers (or taxpayers) fund the development of any software produced by a private company, State employees, or hired consultants.

The use of OSS in ATMS contributes to a healthy ITS market in a number of ways. First, there are no (or small) software acquisition costs. This frees funds for other more productive uses, for example adding enhancements and new capabilities. This benefits taxpayers, DOTs, and consultants. Second, the use of OSS products facilitates cooperation and development between cost-constrained universities and DOTs. Third, the use of OSS (e.g. open-source GIS) enables contributions and enhancements to these open-source projects, from which other users (e.g. other DOTs) benefit. Knowledge and experience gained with OSS products also benefits other organizations. Fourth, DOTs would experience less software product lock-in (see Lock-in, Section 4). This would free funds currently spent on artificially high software licensing costs for more productive uses and encourage competition from vendors and consultants.

In addition to using OSS products on ATMS projects, the creation of an open-source ATMS project would produce further benefits. First, knowledge and software could more easily flow between universities, DOTs, and the private sector. This is expressed well as "the intrinsic parallelism and free idea exchange in OSS has benefits that are not replicable with our current licensing model" (internal Microsoft document [105]). The FHWA is using this approach with their NGSIM project (see pg. 55). Multiple State DOTs, universities, and private sector firms could collaborate and fund an OSS ATMS. As a result, there would likely be an increase in innovation for the same reasons OSS encourages innovation. Second, an OSS ATMS would encourage the development of standards and interoperability. Developed software code would only be reusable by other organizations if it conformed to existing standards. Third, an OSS ATMS would lower entry barriers for commercial firms seeking ATMS-related contracts with DOTs for service, support, and enhancements. This has the potential of increasing the quality and quantity of firms seeking DOT contracts. Fourth, the use of an OSS ATMS eliminates the risk to DOTs of proprietary ATMS vendors going out of business, or not adequately supporting their proprietary system. The demise of software vendors is more likely in small and specialized software markets where vendors have a difficult time financing software development before sales, and customers are reluctant to use products from vendors who might disappear, taking their proprietary products with them.

Use of Standards

The use and development of ATMS standards are crucial for the healthy development of the ATMS market and interoperability (see ITS Standards in Section 7). Because OSS is intrinsically about sharing knowledge, there is inherent motivation to use and develop standards. Any DOT, consultant, or university researcher contributing to an OSS project wants their contributions to be

included in subsequent versions of the product. Nonconforming contributions are less likely to be used by others and less likely to be included. Further, potential enhancements to OSS projects that ignore standards tend to increase lock-in, which is undesirable and to be avoided.

SECTION 6:
OPEN-SOURCE SOFTWARE CONCERNS

This section discusses concerns about OSS: lack of trained staff, inconsistent quality, and support. Other concerns have been reported elsewhere. For example, Mitre reports OSS weaknesses including lack of ownership, difficulty in starting new OSS projects, and less user-friendliness [63]. Table 7 shows the top issues reported in a 2005 study of Linux deployment on servers [7]. Deployment issues that are addressable by commercial markets can be expected to improve. For example, lack of vendor support in 2002 was addressed by the market by 2005. Some OSS deployment concerns are addressed in this section. Benefits provided by OSS and its unique development model must be balanced with a consideration of concerns. An understanding of the OSS development model is crucial for setting expectations. For example, benefits gained from peer review are proportional to the number of individuals involved.

Table 9: Server challenges to deploying Linux [7]

Issue Deploying Linux on Servers	% of Respondents
Technical knowledge of our personnel	35
Compatibility with existing software	33
Problems related to multiple Linux distributions/versions	24
Poor documentation	20
Hardware support problems	20
Poor technical support	17
Lack of widely available rapid development tools	13
Difficult to integrate with existing database server	13
Too new/not widely used	12
Inability to find quality/affordable Linux training	11
Lack of E-commerce components	10
Too complex, too difficult to master	9
Security	6
Reliability	6
No issues encountered	***28***

Enough Trained Staff

A sufficient number of trained staff can be a problem if an organization has no Unix or Linux experience. Nearly all college graduates in computer science or engineering have some amount of experience with Linux, and this trend is expected to continue. A May 2006 search of the Dice technology job site found a nearly equal number of job postings mentioning Unix and Windows experience, and about half that amount mentioning Linux (13751, 13038, and 6936 respectively). Given that Unix skills are somewhat transferable to Linux it should not be difficult to find skilled staff or training for existing staff. For example, the FAA's successful conversion of their Enhanced Traffic Management System to OSS depended on obtaining generous training for a large number of staff at multiple sites (see pg. 44). Sticking with mainstream OSS applications (e.g. MySQL, Eclipse, PHP, etc.) increases the pool of potentially available trained staff.

Inconsistent Quality

Inconsistent quality is a concern for new OSS projects, or projects with fewer users and developers. There is a clear relationship between project code quality and the number of users. Fewer developers means less peer-reviewed code. This also carries over to support, documentation, and security concerns. In a report to address similar issues for the Department of Defense regarding its use of OSS, Mitre Corporation recommended that the DoD create a "Generally Recognized As Safe" list of OSS. They did not recommend banning OSS products not on the list, because of reliance on some specialty open-source tools.

Support

The availability of support has dramatically improved since 2002, when it was a top issue. Both paid and free support are available for hardware, Linux, and mainstream applications such as MySQL. A unique feature of the paid open-source support market is its competitive nature. Because source code is available, entry barriers are low for new firms in the support market. Multiple independent organizations compete in this market. Free support is also available over the Internet from a number of sites (Table 8). Mainstream Linux distributions such as Red Hat and SUSE have both free support through documentation and knowledge bases, and paid support. See Figure 10 for a diagram of Red Hat's support structure. Hardware vendors such as Dell offer preinstalled Red Hat and SUSE. Dell, Red Hat, and Oracle offer a certified Oracle 9i configuration.

Oracle's support for Linux is extensive. Oracle shipped its first database product for Linux in 1998. Linux is expected to be running in 44% of Oracle installations by the end of 2006 [106]. Oracle provides first-line support for Red Hat and SUSE. In addition, all Oracle products are available on Linux.

Table 10: Free Linux support [63]

Vendor	URL
LinuxHelp	http://www.linuxhelp.net/
Linux Documentation Project	www.linuxdoc.org
Linux Support Services	free.linux-support.net
Red Hat Support Links	www.redhat.com/support/docs/tips/urls/urls.html
Novell SUSE Help	http://www.novell.com/services/
News groups (multiple)	comp.os.linux
News groups (multiple)	alt.os.linux

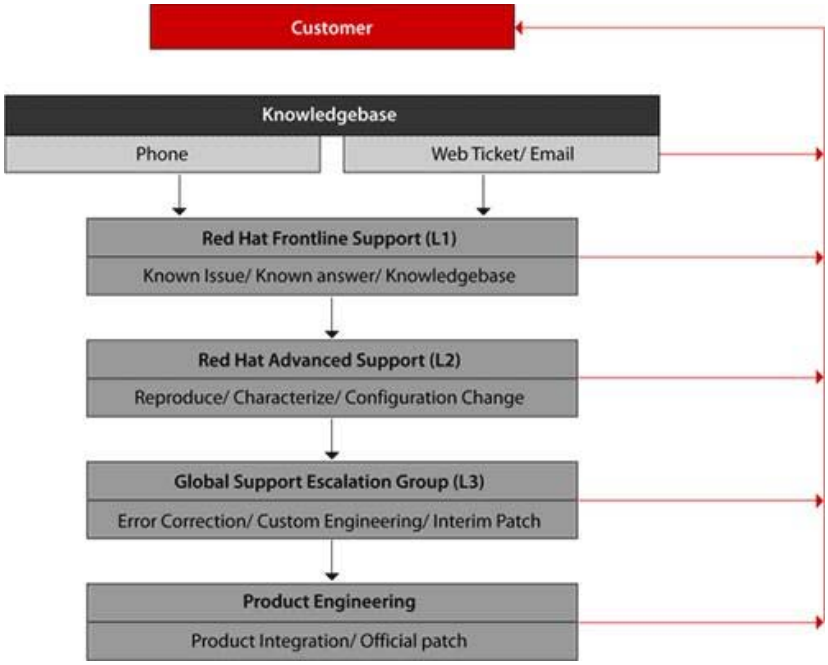

Figure 10: Red Hat support workflow diagram [25]

SECTION 7:
HISTORICAL DEVELOPMENTS IN ATMS

This section discusses historical developments in Advanced Traffic Management Systems (ATMS). ATMS is a primary subfield within the Intelligent Transportation System (ITS) ecosystem. The ATMS view is a top-down management perspective that integrates technology primarily to improve safety and the flow of vehicle traffic. This section briefly discusses ATMS and ITS history, ITS standards, international ITS efforts, goals and benefits of ATMS, and, finally, ATMS trends.

Birth of Intelligent Transportation Systems

In 1956, the *National Interstate and Defense Highways Act* initiated a 35-year $114 billion program that designed and constructed the Interstate highway system. The impact this program had on the United States was enormous. It was mostly complete by 1991, and the era of build-out was over.

Figure 11: 1969 Astro concept car for "systems-controlled interstate highways of the future" [9,28]

In the mid to late 1980s transportation officials from Federal and State governments, the private sector, and universities began a series of informal meetings discussing the future of transportation. This included meetings held by Caltrans in October of 1986 to discuss technology applied to future advanced highways [112]. In June of 1988 in Washington, DC, the group formalized its structure and chose the name Mobility 2000 [93]. The first National Mobility 2000 meeting was held in February of 1989 in San Antonio, Texas, and was organized into four groups: ATMS, Advanced Driver Information Systems (ADIS) (now Advanced Traveler Information Systems or ATIS), Commercial Vehicle Operations (CVO), and Advanced Vehicle Control Systems (AVCS) [93]. In 1990, Mobility 2000 morphed into ITS America, the main ITS advocacy and policy group in the US. The initial name of ITS America was IVHS America—the 1994 change reflects a broader intermodal perspective [111]. With membership consisting of individuals in the public and private sector, ITS America has provided crucial development guidance for subsequent transportation acts and initiatives.

The 1991 *Intermodal Surface Transportation Efficiency Act* (ISTEA) was the first post-build-out transportation act. It initiated a new approach focused on efficiency, intelligence, and

intermodalism. It also had a primary goal of providing "the foundation for the nation to compete in the global economy" [60]. This new mixture of infrastructure and technology was identified as an *Intelligent Transportation System* (ITS) and was the centerpiece of the 1991 ISTEA act. ITS is loosely defined as "the application of computers, communications, and sensor technology to surface transportation" [59].

In 2005 the SAFETEA-LU (Safe, Accountable, Flexible, Efficient Transportation Equity Act: A Legacy for Users) surface transportation spending bill was signed into law.[35] It has the following implications for ITS [84]:

- Provides more funding for ITS and Operations.
- Maintains strong ITS research and development funding.
- Increases focus on congestion relief.
- Puts strong focus on managed lanes and pricing.
- Establishes nationwide requirement for real-time information systems.
- Advances system management and operations.

ITS Standards

Standards are agreements over shared knowledge and are crucial for the healthy development of new markets such as ITS. It is difficult to imagine healthy and efficient markets for electricity, radios, or car tires without government and/or industry standards. "The biggest single retardant in the deployment of ITS is the lack of standards...proprietary systems are secret. It's important to have national, nonproprietary standards" said David J. Hensing, Deputy Executive Director of AASHTO [94]. Standards define what information is open and shared for participants. Inherently a National Intelligent Transportation System would require conceptual and operational integration between private firms, research organizations, State, local, and National governments. To meet these needs the U.S. Department of Transportation initiated development of the *National ITS Architecture* in 1994. The National ITS Architecture provides a common framework for ITS development by defining systems and subsystems and the data that flows between them. For example, it defines eight service areas, one of which is traffic management (ATMS). It further defines 21 market packages within traffic management, for example, Freeway Control is defined as ATMS market package #4 [8]. The 2005 SAFETEA-LU surface transportation bill continues strong support of the National ITS Architecture "to the maximum extent practicable, the national architecture shall promote interoperability among, and efficiency of, intelligent transportation system technologies implemented throughout the United States."[36] Current ITS standards include 81 published, 12 approved, 7 in ballot, and 10 in development [19]. The 1998 Transportation Equity Act for the 21st Century (TEA-21) further required the Department of Transportation to identify and specify "which standards are critical to ensuring national interoperability or critical to the development of other standards and specifying the status of the development of each standard identified" [94]. In 2001, TEA-21 also stipulated denial of federal funding for any ITS project that does not conform to the National ITS

[35] See www.fhwa.dot.gov/safetealu.
[36] See www.fhwa.dot.gov/safetealu, section 5307, subsection A.

Architecture. SAFETEA-LU continues this requirement. ITS projects funded by the Highway Trust Fund must "conform to the national architecture, applicable standards or provisional standards, and protocols."[37]

International ITS Efforts

In Japan, ITS research began in the 1970s with the Comprehensive Automobile Traffic Control System (CACS), which was a centralized route guidance system similar to the concept of the Electronic Route Guidance System (ERGS) in the 1970s in the United States. In 1984, development work on a car navigation system began with the Road/Automobile Communication System (RACS) and the Advanced Mobile Traffic Information and Communications System (AMTICS). Presently the ITS program is part of the generalized National IT program called the Advanced Information and Telecommunications Society, which has a national goal for Japan "to become the world's most advanced IT nation" [17]. ITS Japan, also known as Vehicle, Road and Traffic Intelligence Society or VERTIS, is analogous to ITS America, promoting ITS research and development, standards development, deployment, and public/private cooperation.

In Europe, the PROMETHEUS program (Program for European Traffic with Highest Efficiency and Unprecedented Safety) was started in 1986 and funded by automobile companies, research institutions, and governments to produce a common technology platform [97]. The emphasis was on sophisticated in-vehicle technology. In 1989, the DRIVE program (Dedicated Road Infrastructures for Vehicle safety in Europe) was started with the goals of improving traffic efficiency and safety. In 1991, ERTICO (European Road Transport Telematics Implementation Coordination Organization) was created to develop policies and strategies, and coordinate numerous ITS projects. ERTICO is similar to ITS America with an overall goal to enhance European competitiveness with ITS and market-driven investment.

ATMS Goals and Benefits

ATMS provides a top-down perspective that integrates technology primarily to improve the flow of traffic. Real-time infrastructure data from cameras, speed sensors, etc., flows into a TMC where they are integrated and processed (e.g. for incident detection), and may result in actions taken (e.g. traffic routing, CMS message updates) with the goal of improving traffic flow. The National ITS Architecture defines the following primary goals and associated metrics for ITS [6]:

- Increase transportation system efficiency
- Enhance mobility
- Improve safety
- Reduce fuel consumption and environmental cost
- Increase economic productivity
- Create an environment for an ITS market

[37] See the SAFETEA-LU Public Law, www.fhwa.dot.gov/safetealu, section 5307, subsection C.

It also defines a detailed goals matrix for the 21 market packages within the traffic management service area. This matrix is reproduced in Table 11.

Looking forward, so-called ITS-4 technologies have been identified as sensing, communication, and processing speed, combined to produce real-time results [104]. Improvements will enable increasingly sophisticated ATMS applications.

ITS/ATMS Milestones

- 1988 Mobility 2000
- 1990 Formation of ITS America
- 1991 End of build-out era
- 1991 ISTEA (Intermodal Surface Transportation Efficiency Act)
- 1994 National ITS Architecture
- 1997 National Automated Highway Systems Consortium Demonstration
- 1998 TEA-21, Transportation Equity Act for the 21st Century
- 2002/3 Ten-Year ITS Plan
- 2005 SAFETEA-LU, Safe, Accountable, Flexible, Efficient Transportation Equity Act: A Legacy for Users

Commodity IP video cameras are a good example and are referenced in several case studies in this report. IP video cameras use sophisticated compression, simplify and reduce cabling costs, and enable low-cost public distribution of images over the Internet. Expanding camera intelligence will only increase, e.g. automatic scaling of image resolution and frame rate using built-in intelligence. One can easily imagine automatic detection and notification of emergency traffic situations. Commercial units that provide gunshot detection integrated with video in urban environments are available, and research sponsored by the U.S. Army is currently investigating seismic and olfactory sensors [26].

Trends inside the ATMS market are continuing to push change. As ITS build-out continues into lower-density population areas, cost-sensitivity increases—the primary obstacle to implementing ATMS is no longer technology, it is cost. This is exemplified by California's hesitation to implement its existing ATMS system in all twelve districts. It is significant that the more cost-sensitive states are developing the most innovative low-cost ATMS implementations (see case studies, pg. 39). A trend directly related to lowering costs is the increased use of standards. Increased ITS interoperability requirements are also driving standardization efforts. Finally, increasing Homeland Security requirements appear to be a long-term trend driving some ATMS functional requirements such as video surveillance.

Table 11: National ITS Architecture ATMS benefits matrix [5]

Market Package Name	Increase Transportation System Efficiency	Enhance Mobility	Improve Safety	Reduce Environmental Cost	Increase Economic Productivity	Create Environment for an ITS Market
Network Surveillance	√	√		√		√
Probe Surveillance	√	√		√		√√
Surface Street Control	√√	√√√	√√	√√		√
Freeway Control	√√	√√√	√	√√		√
HOV Lane Management	√	√√		√		√
Traffic Information Dissemination	√√	√		√		√
Regional Traffic Control	√√√	√√√	√√	√√√		√
Traffic Incident Management System	√√	√√	√√	√√√		√
Traffic Forecast and Demand Management	√√	√√				√
Electronic Toll Collection					√√	√
Emissions Monitoring and Management				√√√		√√
Virtual TMC and Smart Probe Data	√	√		√	√	√
Standard Railroad Grade Crossing			√√√			√
Advanced Railroad Grade Crossing			√√√			√
Railroad Operations Coordination	√	√		√		√
Parking Facility Management	√√			√	√	
Regional Parking Management	√√	√		√		
Reversible Lane Management	√√	√		√		
Speed Monitoring	√√	√	√√√			√
Drawbridge Management	√√	√√	√		√	
Roadway Closure Management	√	√√		√		√

Key—Satisfies goals: √ Marginally, √√ Almost every aspect, √√√ Completely

ATMS Trends

Trends in ATMS and ITS are driven both internally by customer needs and externally by advances in technology. Semiconductor process technology continues to lower power requirements, reduce chip size, and double performance about every eighteen months. This is simultaneously pushing decentralization of processing intelligence and commoditization of

sensors, processors, and communication. These are combined to produce real-time results, and have been described as next-generation "ITS-4" technologies [96]. The commoditization of hardware parallels the commoditization of software in the form of OSS.

The Advanced Transportation Controller (ATC) standardization effort is taking advantage of these trends by using commodity hardware and software: "as the current trend continues towards distributing more of the intelligence of ITS out closer to the field, there is an increasing demand for more and more capable field deployable devices. This hardware must run more sophisticated applications software and operate in modern networking environments. The ATC Controller is intended to address these needs" [11]. The ATC standard identifies software costs as "one of the largest component costs of today's Intelligent Transportation Systems." The standard seeks to addresses this concern with the inclusion of the OSS operating system Linux as part of the standard.

SECTION 8:
CASE STUDIES OF ITS SOFTWARE PROJECTS USING OPEN-SOURCE SOFTWARE

This section discusses case studies of closed-source ITS software projects that *use* open-source software, which are distinct from projects that *are* open-source projects (see the following section for this latter category). Projects discussed here use OSS to implement software applications that are themselves closed-source. For example, they might use MySQL, Linux, Apache, etc., along with developed software code to build the application. The developed source code is not shared with other organizations. The majority of case studies fall into this category. This is often the first step an organization takes towards the open-source approach, replacing proprietary software tools with open-source counterparts. Project organization is more or less unchanged compared with the proprietary approach. Subsequent sections cover projects that are more hardware-oriented. The focus here is on ATMS projects; however, other related projects are discussed. Each case study strives to answer who, why, what, and how, with respect to OSS.

It is significant that many of the projects listed here are explicitly low-cost. Most of the projects are associated with research organizations. This is significant because research organizations tend to be more cost-constrained than State DOTs. It seems clear that the increased use of OSS by State DOTs will increase the flow of innovation, ideas, and new developments between DOTs and research institutions, benefiting both, along with the traveling public. Several individuals involved in the following case studies expressed interest in sharing their work with others in the form of an open-source project. There is clearly an unmet need and interest among research institutions and states for sharing knowledge and software code.

Virginia Department of Transportation Web-based Congestion Monitoring ATMS

The Virginia Web-based Congestion Monitoring ATMS project is implemented using OSS. The Smart Travel Lab is an ITS research group at the University of Virginia [104]. In September of 2003 they completed the ATMS project "Web-based Congestion Monitoring Map for NOVA Smart Traffic Signal System" for the Virginia Department of Transportation [86]. The project had three objectives:

- Develop a real-time congestion metric for signalized intersections.
- Provide Internet-based intersection-level maps indicating congestion levels.
- Provide congestion trend data.

The developed congestion metric used an intersection volume-to-capacity ratio. The congestion maps were used to identify intersections with poor signal-timing performance for reoptimization.

The web-based application presents users with a clickable map of Virginia's road network (Figure 12, pg. 40). The system provides network- and intersection-level maps with real-time and historic traffic data and covers three Virginia DOT (VDOT) TMCs and 1000+ signalized intersections. Real-time traffic data is updated on 15-minute intervals. The system is designed for both TMC ATMS personnel and ATIS users. Real-time traffic data is stored in a MySQL

database running on the Linux operating system. Web-based mapping is provided by ESRI's ArcIMS. Daily traffic data is loaded into an Oracle 8i database each evening (Figure 13, pg. 40).

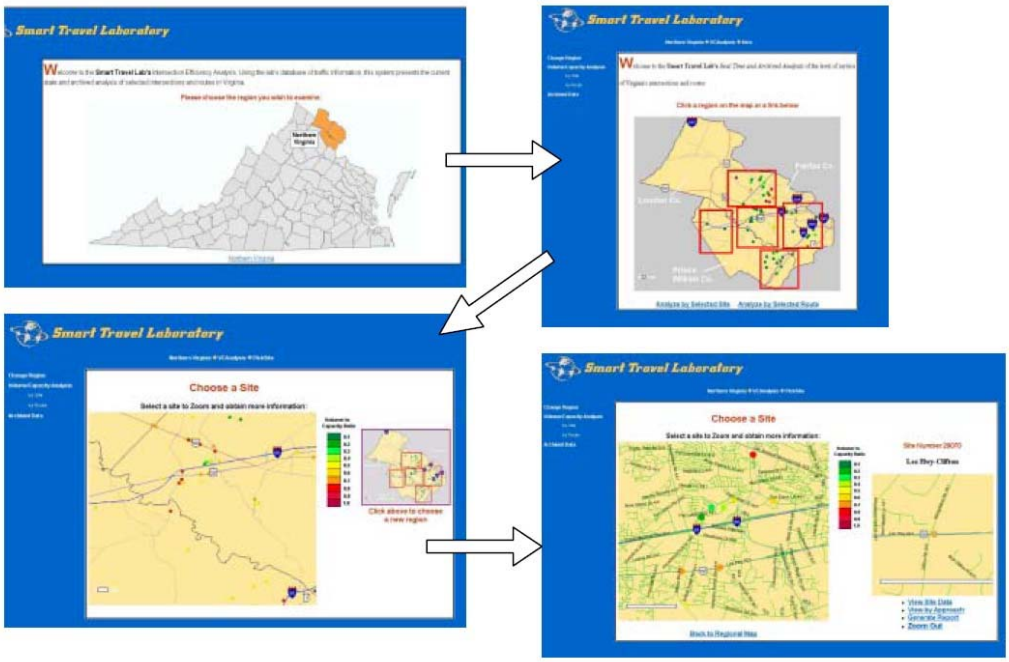

Figure 12: Web-based congestion maps from Virginia ATMS [86]

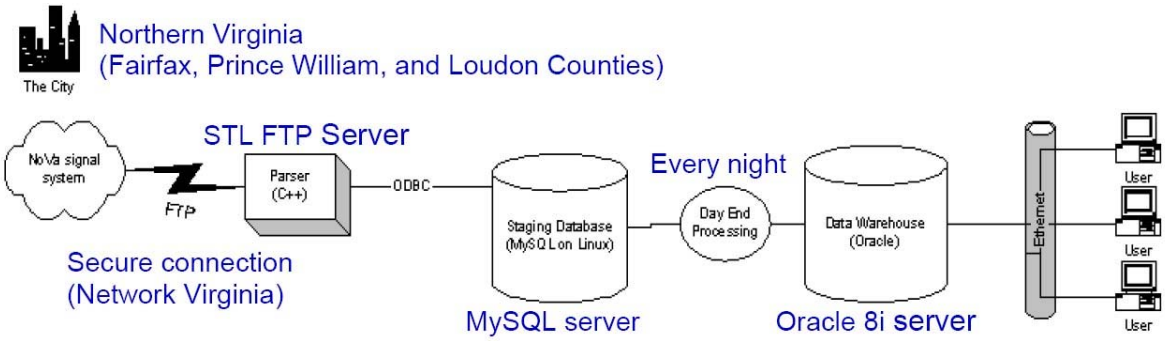

Figure 13: Architecture of Virginia ATMS Web-based Congestion Monitoring [86]

The system uses the existing Oracle database for historical traffic data and MySQL for real-time data. MySQL was used to lower the load on the existing Oracle database and to reduce database licensing fees. MySQL was also used because it was "free, reliable, easy to use, has large user base" [85]. A disadvantage reported was that this staged architecture requires an extra step, presumably because of the added complexity of moving data between two databases [85].

Oklahoma Department of Transportation Statewide Distributed Low-Cost ITS

The Oklahoma Statewide Distributed Low-cost ITS project is implemented using OSS. The Oklahoma Department of Transportation is currently developing a Statewide ITS with a novel

distributed architecture [57]. The University of Oklahoma is developing the system. Cost was a major concern for ITS deployment in Oklahoma. To address this issue, a low–cost and distributed ITS is currently being deployed. A guiding design philosophy is that "any console should be able to control any system resource at any time." This enables the elimination of an expensive centralized TMC. TMC operators are connected by a peer-to-peer network into a virtual, geographically-distributed, and fault-tolerant TMC.

System-wide functional requirements specify that the system must handle incident management, work zone traffic management, weather information monitoring, critical infrastructure monitoring, commercial vehicle operations, and public dissemination of information. Security privilege levels are used to control access to resources. Design requirements specify that an ITS console user operating anywhere in the State should be able to:

- Control any ITS device at any time.
- View video from any system camera.
- Post messages to all ITS consoles Statewide.
- Forward information (warnings, images, messages, video) to the ATIS and 511 systems.

The system is constructed with CCTV cameras, IP web cameras, CMSs, Remote Traffic Detectors (RTDs), and Remote Weather Stations (RWSs). Various codecs were tested (MPEG-2, MPEG-4, MJPEG) and ultimately MPEG-2 was selected. Figure 14 (pg. 42) shows the network architecture diagram.

The ITS console runs on commodity x86 hardware. To control costs "a concerted effort has been made to base the software architecture on open-source and public domain packages where possible in order to avoid costly and recurrent software licensing fees." The ITS console is constructed with Microsoft Windows XP and the following open-source products:

- Apache web server (www.apache.org),
- PHP interpreter (www.php.net),
- MySQL database (www.mysql.com), and
- MapServer GIS (mapserver.gis.umn.edu).

The Apache web server is running locally on the ITS console machine. Access to applications is through the web browser. Other development tools include Microsoft Visual Basic, Visual C++, and JavaScript. The ITS console displays analog and digital video, and controls CCTV cameras, CMSs, RTDs, RWSs, and other sensors. A message queue and database are used to control device access (e.g. to CMS), event logging, and user messages.

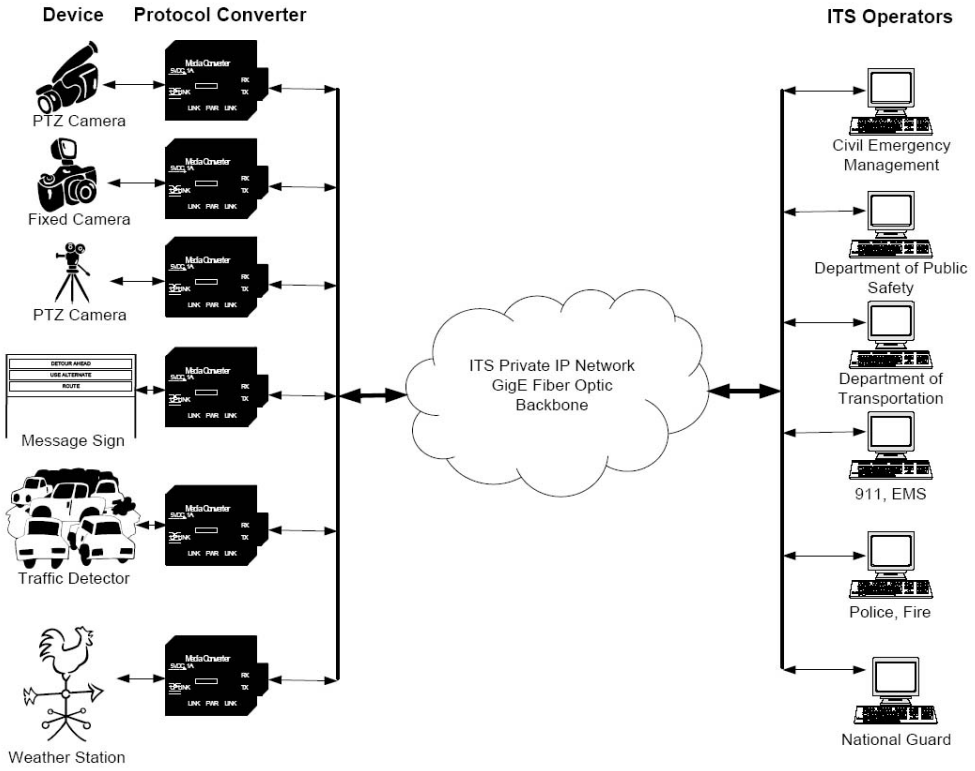

Figure 14: Oklahoma ITS Distributed IP Network Architecture [57]

Oklahoma Department of Transportation SAFE-T Accident Analysis System

The Oklahoma SAFE-T (Statewide Analysis For Engineering and Technology) Accident Analysis System is a project that uses OSS [99]. The Oklahoma Department of Transportation is currently developing the system to provide traffic engineering decision-support. It is presently being used by the Oklahoma DOT, the Oklahoma Highway Patrol, and municipal and local traffic engineers statewide. The goal is to reduce crashes using automated traffic analysis of highway enhancements and construction. The system uses the MySQL database and MapServer GIS products [56]. The University of Oklahoma is developing the system.

Oklahoma Department of Transportation ATIS System

The Oklahoma ATIS (Advanced Traveler Information System) System is a project that uses OSS. The Oklahoma DOT is funding the project, which is a web-based system [103], and the University of Oklahoma is developing the system. The application will provide real-time traffic data for Oklahoma City and will be used by TMC personnel and the traveling public. The system will provide real-time average speed, road conditions, construction information, and web-cam images. The system is implemented with the open-source database MySQL, the Apache web server, and the MapServer GIS [56].

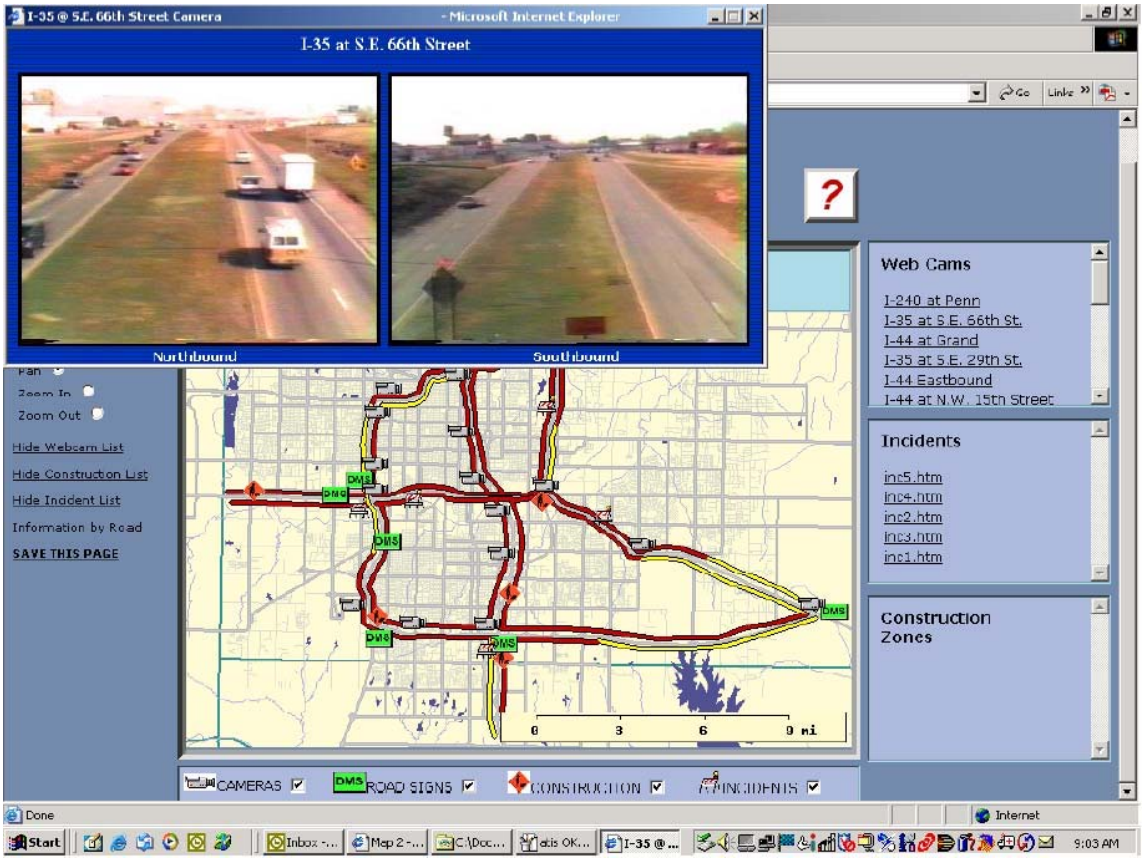

Figure 15: Oklahoma Web-based ATIS [18]

Minnesota Department of Transportation IRIS Intelligent Roadway Information System

The Minnesota IRIS Intelligent Roadway Information System is a project that uses OSS. The Minnesota Departmental of Transportation (Mn/DOT) Regional Transportation Management Center's (RTMC) goals are to improve safety and reduce congestion using "cutting-edge technology, progressive programs, and real-time information delivery systems" [21]. The RTMC is co-located in a unified command center with the State Highway Patrol, Mn/DOT's Metro District Maintenance Dispatch, and the Office of Traffic, Security and Operations. The RTMC's ATMS operations software (IRIS) provides ramp metering, incident detection, and CMS control. Mn/DOT's RTMC implements all of their software projects using OSS, and is interested in collaborative partnerships with other DOTs or organizations. The RTMC plans to continue using OSS and is pleased with the results [64].

Minnesota's IRIS ATMS system uses Linux and the open-source database PostgreSQL, and is implemented with Java. The client displays a GUI map and is supported on Linux and Windows. Commodity server hardware is used, including PCI-based serial communication with

field elements. Other RTMC projects are using OSS components including Apache, Tomcat, Batik, Hibernate, Struts, Velocity, Ant, Eclipse, Python, and Mercurial.[38]

Key benefits of the OSS approach are reported to be low initial and recurring costs, and the ability to customize. Software stability and reliability are also reported to be high. It was also reported that moving from the "tried-and-tested" OSS projects to the "cutting-edge" OSS projects tends to result in a higher number of unanticipated problems. Working with "cutting-edge" OSS projects required developer time to resolve issues, reported to be similar to working with a beta version of commercial software.

FAA Real-time Enhanced Air Traffic Management System

The FAA Real-time Enhanced Air Traffic Management System is a project that uses OSS. This report is primarily concerned with surface transportation. However, the Federal Aviation Administration's (FAA) recent technical refresh of their Enhanced Traffic Management System (ETMS) offers insight into what is possible using commodity hardware and OSS. The primary goal of the FAA's ETMS system is to enable air traffic managers to control air traffic so that system capacity is not exceeded. ETMS is used nationwide to monitor real time air traffic volume, surges, and gaps. It integrates five types of data: geographic maps, traffic situation data, alert data, flight list and count data, and weather data [13].

During the technical refresh in 2004 and 2005, the FAA replaced all proprietary RISC workstations with commodity workstations running Linux, reducing per-workstation costs from $25,000 to $3,000. The ETMS system has 1,200 workstations scattered across 100 sites. As part of the technical refresh, software consisting of 1.5 million lines of code was ported to Linux [16]. The initial cost estimate for the technical refresh of the original proprietary RISC system was $25 million; however, the technical refresh, using commodity hardware and Linux, was achieved with $10 million. In addition, performance improved substantially [13].

Prior to the ETMS refresh, Linux was used in the FAA's Common Automated Radar Terminal System (ARTS) [35]. The ARTS system was first used in Atlanta in 1964 and was expanded and upgraded over time [12]. By 1973, the system was used nationwide. In the 1980s, the software was ported to C. The system was ported to Linux in 2000 for testing purposes because the proprietary production hardware was so expensive. The FAA began production use of Common ARTS running on Linux in 2003 on a non-critical system (Figure 16). As the FAA gained experience with Linux and commodity hardware, their Linux use has expanded.

[38] See www.apache.org, tomcat.apache.org, xmlgraphics.apache.org/batik, www hibernate.org, struts.apache.org, jakarta.apache.org/velocity, ant.apache.org, www.eclipse.org, www.python.org, www.selenic.com/mercurial.

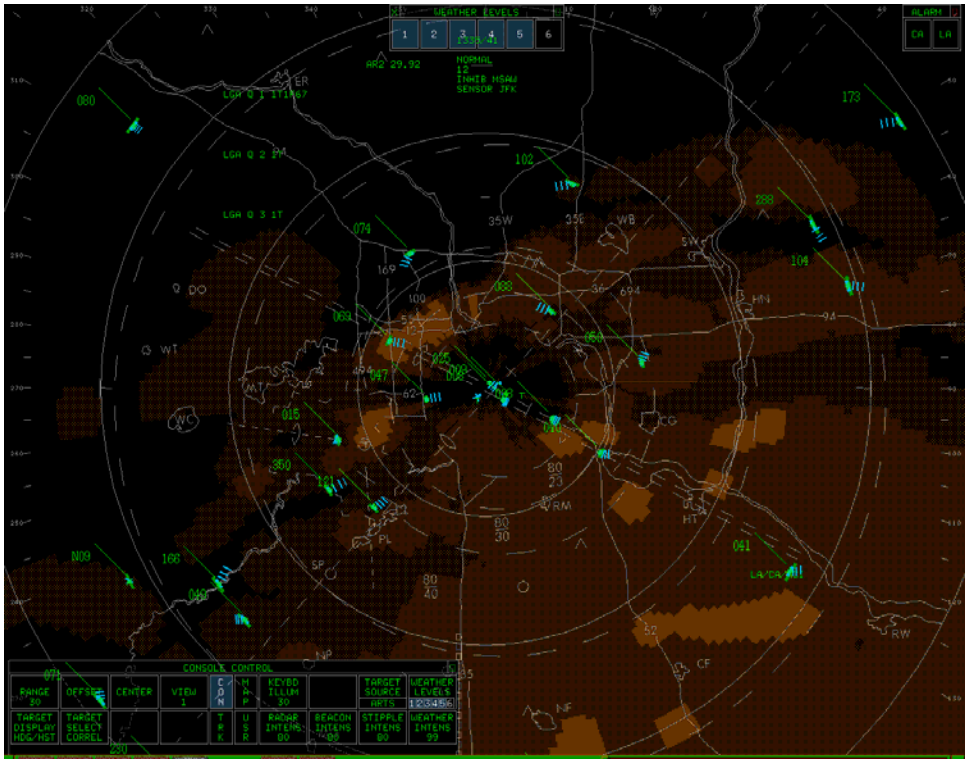

Figure 16: Linux ARTS screenshot [35]

U.S. DOT Weather-Related Road Hazards Assessment and Monitoring System

The Weather-Related Road Hazards Assessment and Monitoring System is a project that uses OSS. The National Consortium on Remote Sensing in Transportation (NCRST) funded the research demonstration project Weather-Related Road Hazards Assessment and Monitoring System (WRRHAMS) [20]. The project goal was to develop a web-based application for "for identifying and visualizing transportation infrastructure locations in danger of having been damaged by precipitation events" for rural unpaved roads in New Mexico [114]. The system used near-real-time NEXRAD (Next Generation Radar) Doppler radar precipitation data combined with local soil, vegetation, and terrain data. The output product is a map indicating areas with a high probability of flash flood damage to rural roads. The system started operating in February of 2003. Operation ceased due to funding problems. The system was implemented using Linux and the open-source GRASS (Geographic Resources Analysis Support System) GIS.[39]

[39] See grass.itc.it.

Figure 17: WRRHAMS web application screen shot [114]

Pennsylvania State Hourly Mesonet

The Pennsylvania Hourly Mesonet project uses OSS—it is a centralized repository of historic and real-time weather station data developed and maintained by the Pennsylvania State Climatologist at the Pennsylvania State University, Department of Meteorology[40] . Data is received from approximately 160 networked weather stations (Figure 18) operated by the FAA, PennDOT (Pennsylvania Department of Transportation), and PADEP (Pennsylvania Department of Environmental Protection) [102]. Minimally these stations report hourly temperature (Figure 19), wind direction, and wind speed. Plans are underway to incorporate the approximately 125 National Weather Service (NWS) Cooperative Observer Program (COOP) stations. Weather station data is processed hourly. Derived information includes wind streamlines, temperatures, dew point contours, and time series displays of information. Roadway Weather Information System (RWIS) sensors are used to monitor roadway temperature, ground condition (e.g. icy, wet, snow cover), and average vehicle speeds. New data sources are being added as their data passes a certain level of quality assurance. For example, the Citizen's Weather Observation Program (CWOP) has more than 100 hourly reporting sites in Pennsylvania.

[40] See the Pennsylvania Mesonet site at pasc.met.psu.edu/MESONET and www.met.psu.edu.

Hourly Mesonet data is stored in a MySQL database. The OSS Perl scripting language[41] is used to read flat data files received from sensors and to insert the results into the MySQL database. The Mesonet web site uses the Linux operating system and Apache web server. In general, the Pennsylvania State University Department of Meteorology uses Java, PHP, Perl, Linux, MySQL, Flash (a proprietary product from Adobe, Inc.), and Red Hat Enterprise Linux [30]. Red Hat was selected for its quality and ongoing support. Satisfaction is reported to be high. Software is developed by a combination of undergraduate and graduate students and a few full-time research assistants.

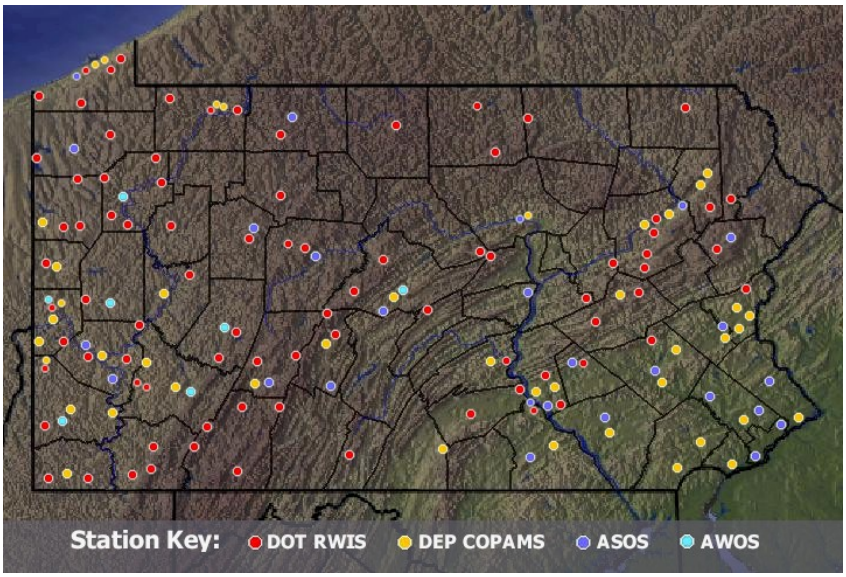

Figure 18: Pennsylvania Mesonet weather station locations

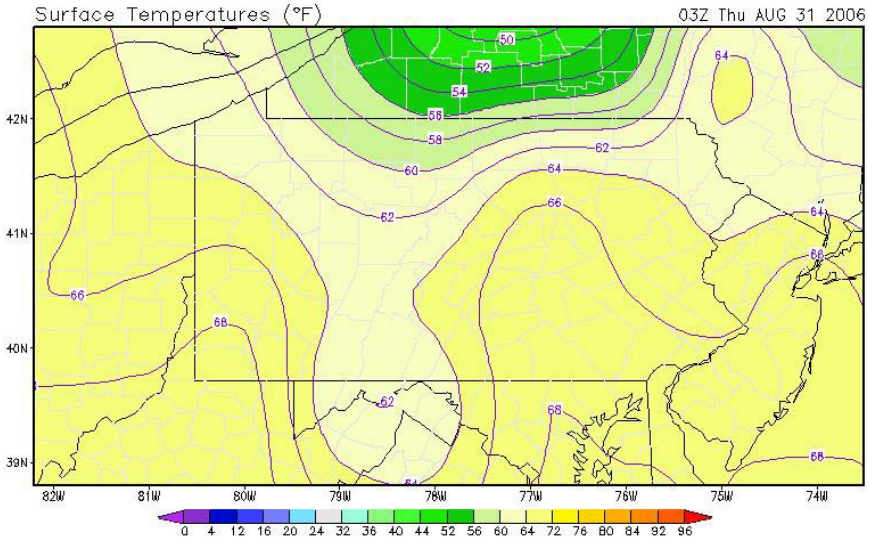

Figure 19: Pennsylvania Mesonet real-time surface temperature analysis

[41] For Perl, see www.perl.org

Los Angeles County Regional ITS Integration Project

The Los Angeles County RIITS (Regional Integration of Intelligent Transportation Systems) project is implemented with OSS, and is sponsored by the Los Angeles County Metropolitan Transportation Authority[42]. Its primary goal is to increase the effectiveness of investments already made in electronic information systems and transportation systems through the real-time exchange of traffic and transportation data. Agencies involved are Caltrans, the California Highway Patrol, the City of Los Angeles Department of Transportation, and the Los Angeles County Metropolitan Transportation Authority (see Table 12 for integration data types). Examples of benefits expected from information sharing include [45]:

- Help dispatchers identify congested routes and identify alternatives;
- Allow law enforcement and emergency services to track GPS-equipped transit vehicles that require assistance;
- Provide video images of freeway incidents to multiple agencies;
- Inform transit operators of expected transit vehicle arrival times;
- Increase airport and port operators' understanding of road conditions affecting passenger and truck arrivals.

Table 12: RIITS data available via web services [71]

Data Type	Agency Providing Data	Update Rates
Congestion – Freeway Inventory	Caltrans D7	Daily/Midnight
Congestion – Freeway Real-time	Caltrans D7	Every 1 Minute
Congestion – Arterial Inventory	LADOT	Daily/Midnight
Congestion – Arterial Real-time	LADOT	Every 1 Minute
Congestion – Arterial Inventory	Caltrans D7	Quarterly
Congestion – Arterial Real-time	Caltrans D7	Every 1 Minute
Event	CHP	Every 1 Minute
Event	Caltrans D7	Every 1 Minute
Bus Routes – Inventory	MTA – Metro	Quarterly
Bus Real-time Locations	MTA - Metro	Once per 2 Minutes
Train Routes – Inventory	MTA – Metro	Quarterly
Train Real-time Locations	MTA – Metro	Every 1 Minute
CMS Inventory	Caltrans D7	Daily/Midnight
CMS Real-Time	Caltrans D7	Every 1 Minute
CCTV Inventory	Caltrans D7	Quarterly
CCTV Snapshots	Caltrans D7	Every 1 Minute

The RIITS project uses both OSS and COTS software. It uses an innovative approach with extensive use of web services and XML for data integration. It is a web- and Java-based

[42] See the RIITS web site at www.riits.net.

application. Figure 20 provides a screenshot. Open-source software used on the project includes Red Hat Fedora Linux, JDK 1.4 (Java Development Kit), Apache Tomcat JSP (Java Server Pages) Server, and the Apache Axis SOAP provider [70]. The project was developed by Delcan Inc., who reported decreasing interest among government organizations in using high-cost proprietary software products to build ATMS applications [46].

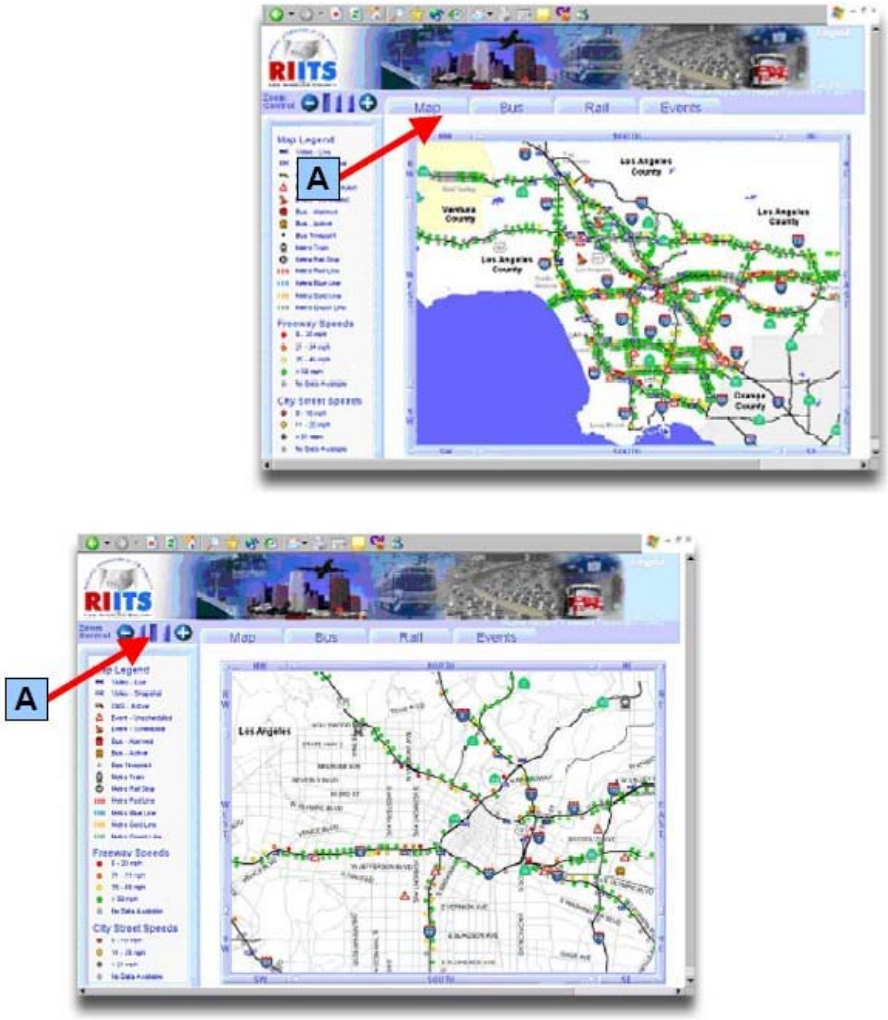

Figure 20: RIITS maps at different zoom levels [72]

University of Maryland RITIS System

The University of Maryland CATT Laboratory (Center for Advanced Transportation Technology) is developing RITIS (Regional Integrated Transportation Information System) using open-source software.[43] The goal of RITIS is to improve transportation efficiency, safety, and security by integrating existing real-time and archival regional transportation data from Virginia, Maryland, and the District of Columbia, among others (see Figure 21). The system collects and disseminates data in a standardized format to a wide variety of users, providing real-time data for traveler information and operations. Archived data is available for research,

[43]For the University of Maryland CATT Laboratory, see www.cattlab.umd.edu.

planning, operational needs, and performance monitoring. Traffic and incident data is archived indefinitely. A website suitable for PDA use is also available. Prototype screen shots are shown in Figure 22 and Figure 23.

RITIS is built with OSS, including the Apache HTTP Server, PHP, MapServer, PostgreSQL, the JBoss Application Server, MySQL, JDOM,[44] and Log4J.[45] Traffic detector data and incident data are available in XML. RITIS data is available to other systems via XML and SOAP, JMS (Java Messaging Service), and others. In general, the CATT Laboratory makes extensive use of OSS [48], motivated in part by the desire to make CATT developed software projects affordable to cost-sensitive agencies such as county traffic management centers. CATT reports that their overall experience with OSS is positive. They have found that OSS software quality varies widely between projects, and that the best software seems to be produced by organizations and foundations which oversee projects and provide direction. They noted that the Apache HTTP Server is highly reliable, easy to configure, easy to maintain, well documented, and supported by a vast user community. They also noted some quirks, such as the need to recompile for SSL support. CATT reported that the JBoss Application Server is complex to configure and suffers from a "severe lack of documentation and user support." CATT also noted that a number of other J2EE servers are available such as JOnAS,[46] and GlassFish,[47] among others.

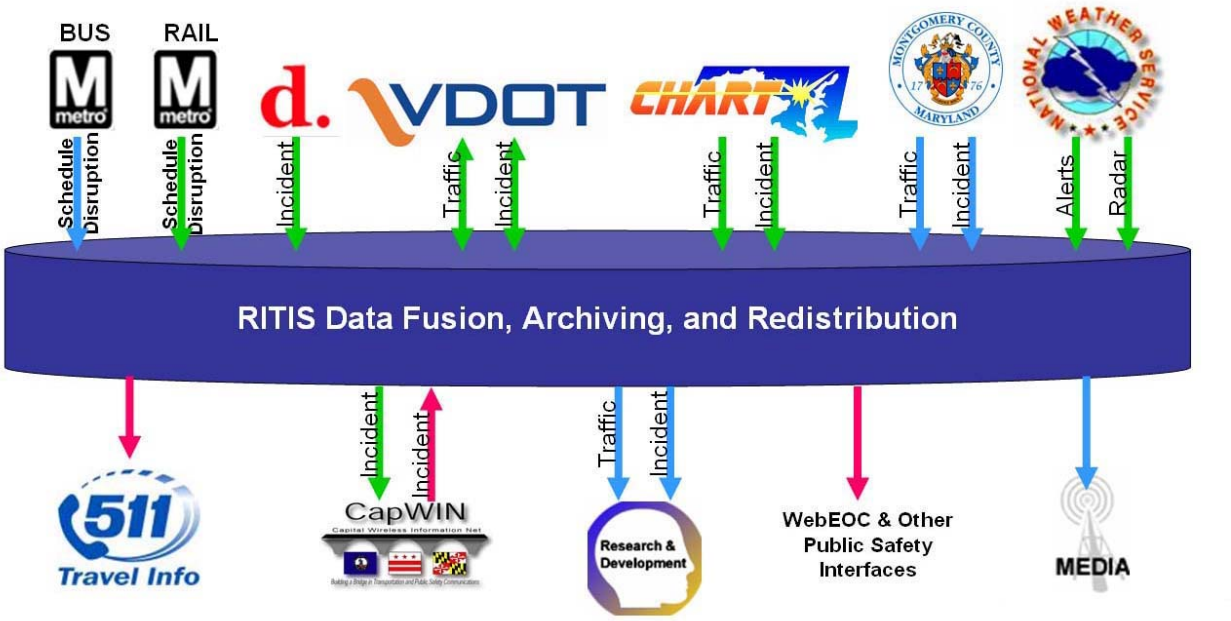

Figure 21: University of Maryland CATT Laboratory RITIS Data Distribution [82]

[44] For JDOM, see www.jdom.org.
[45] For Log4J, see logging.apache.org.
[46] For the JOnAS Java J2EE server, see jonas.objectweb.org.
[47] For the GlassFish Java J2EE server, see glassfish.dev.java net.

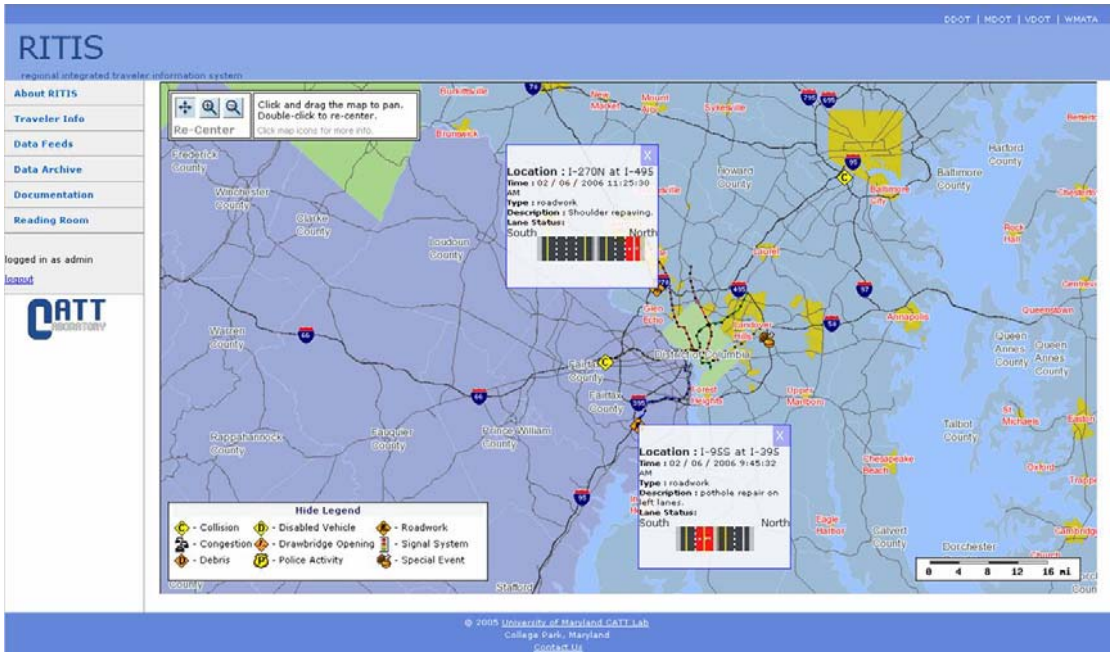

Figure 22: University of Maryland CATT Laboratory RITIS Prototype Screen [82]

Figure 23: University of Maryland CATT Laboratory RITIS PDA Prototype Screens [82]

Virginia Department of Transportation Incident Management System

The University of Maryland CATT Laboratory (Center for Advanced Transportation Technology) [48] is developing an incident management system using open-source software for the Virginia Department of Transportation (VDOT).[49] This incident management system was rapidly designed and developed to meet VDOT's immediate TMC software needs until a full-featured solution can be developed. It provides data collection, performance measures, mapping, and other features. It is designed to be maintainable, customizable, and easy to setup. Other organizations have expressed interest in adopting it. CATT's Incident Management System was developed using the Apache HTTP Server, PHP, MapServer, PostgreSQL, AMFPHP[50], and the Wildfire Server.[51] The CATT Laboratory makes extensive use of OSS. See pg. 49 for another CATT case study and additional information.

[48]For the University of Maryland CATT Laboratory, see www.cattlab.umd.edu.
[49]For the Virginia Department of Transportation, see www.virginiadot.org.
[50] For AMFPHP, see www.amfphp.org.
[51] For the Wildfire Server, see www.jivesoftware.org/wildfire.

SECTION 9:
CASE STUDIES OF ITS OPEN-SOURCE SOFTWARE PROJECTS

This section discusses case studies of ITS software projects that *are* open-source software projects, as distinct from those discussed in the prior section, which are closed-source software projects that *use* open-source software. Projects discussed here are explicitly organized to share project artifacts: source code, datasets, test results, etc. with a community of users who may contribute to the project in some form. The community benefits from these contributions. Projects in this category use some type of OSS license to grant users access to project artifacts. See Appendix D for information on open-source licenses (pg. 89). Subsequent sections discuss ITS projects that are more hardware-oriented. The focus here is on ATMS projects; however, other related projects are discussed. Each case study strives to answer who, why, what, and how, with respect to OSS.

Massachusetts Institute of Technology MITSIMLab Traffic Simulator

The MIT Open-Source MITSIMLab Traffic Simulator is an open-source project that is also implemented using OSS. MITSIMLab is a traffic simulation package developed by MIT's Intelligent Transportation Systems Program and is used to evaluate ATMS and ATIS systems [75]. MITSIMLab uses a license that requires modifications and enhancements to be posted in the public domain and provides source code for downloading. Technical support for the open-source version of MITSIMLab is provided by a public newsgroup, which has 45 members, as of May 2006. MITSIMLab's functionality can be organized into three categories (Figure 24, pg. 54) which are discussed below:

- Microscopic traffic simulation (MITSIM),
- Traffic management simulation (TMS), and
- Graphical user interface (GUI)

The MITSIM component models the physical world, which includes the transportation network and traffic. The transportation network is defined by lanes, links, nodes, traffic controls, and surveillance devices. Statistically selected predefined routes are used to simulate traffic. Each vehicle is simulated using desired speed, aggressiveness, vehicle characteristics, a car-following model, a lane-changing model, responses to signals, speed limits, accidents, and toll booths [74].

The TMS component simulates the traffic control system. Simulation parameters include ramp control, lane control signs, variable speed limit signs, tunnel entrance signs, intersection control, changeable message signs, and in-vehicle route guidance.

The GUI component is used for debugging and viewing network performance (Figure 25).

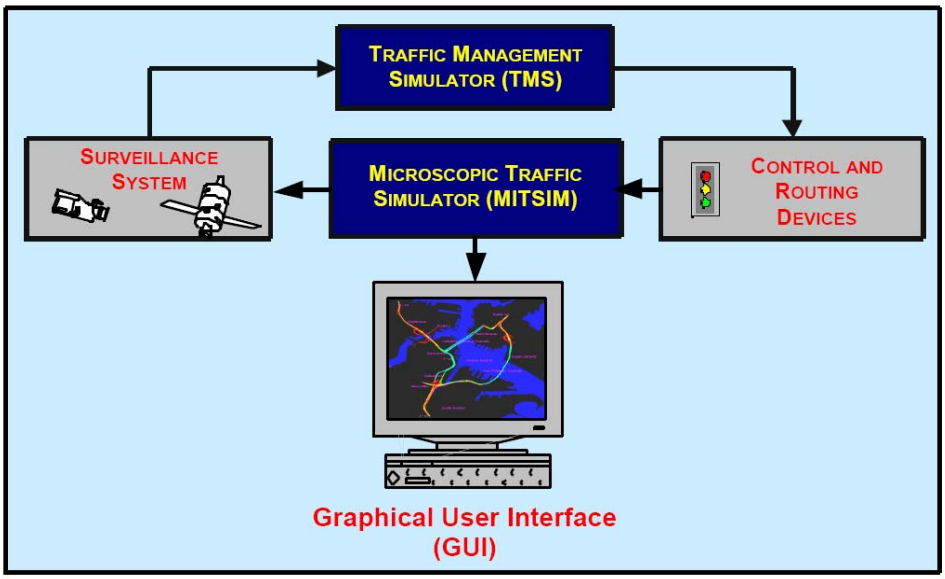

Figure 24: MITSIMLab components [74]

Figure 25: MITSIMLab interface [36]

The city of Stockholm Sweden evaluated MITSIMLab and five other simulation products in a detailed comparison [31,36]. MITSIMLab's simulation abilities were judged superior overall. However, a proprietary competitor (AIMSUN2, Advanced Interactive Microscopic Simulator for

Urban and Non-Urban Networks[52]) was judged to have more complete documentation and better stability.

Federal Highway Administration Next Generation Simulation

The Next Generation Simulation project uses the open-source approach by releasing some project products using an open-source license and encouraging community involvement. The Federal Highway Administration's Next Generation Simulation program (NGSIM) is a traffic simulation research community that is openly sharing data sets, documentation, and algorithms with the transportation community [22]. Program goals are to improve traffic simulation tools, promote the use of simulation, and ensure the accuracy and trust of traffic simulation tools by providing validated simulation results. To achieve these goals FHWA is acting as a "Market Facilitator" to promote the development of the commercial traffic simulation market. Application development will be left to the private sector and FHWA explicitly will not compete in the market. In addition, FHWA will establish standard NGSIM data formats, a web site (www.ngsim.fhwa.dot.gov), and repository. It is notable that this innovative program is sharing knowledge with the goal of establishing a healthy commercial market in traffic simulation.

The NGSIM program uses a Creative Commons copyright license, which is similar to the GNU GPL license.[53] It stipulates that:[54]

- You must give the FHWA and the NGSIM program credit for the data;
- For any reuse or distribution, you must make clear to others that any such reuse or distribution is subject to the license terms of this work; and
- Any of these conditions can be waived if you obtain written permission from the FHWA.

Creative Commons is a non-profit organization started in 2001 by Lawrence Lessig, a law professor at Stanford. Creative Commons provides a variety of free copyright licenses to copyright holders through their web site.[55] Their goal is to facilitate sharing of information.

U.S. DOT Open-Source TEXAS Intersection Simulation Model

The TEXAS intersection simulation model is an open-source project. In May of 2003 the U.S. Department of Transportation requested enhancements to the TEXAS (Traffic EXperimental and Analysis Simulation) microscopic single-intersection simulation model [3]. The TEXAS simulator models sub-microscopic behavior of vehicles as they pass through intersections and mix with other traffic flows. The enhancements include writing a new Java-based user interface, and interfacing with the simulation engine which is written in Fortran 90/95. All of the source code for this project will be copyrighted using the Free Software Foundation's GPL [54]. The project solicitation states "the vendor would be expected to make

[52] See www.aimsum.com.
[53] See creativecommons.org/licenses/by/2.0/ for the specific license used.
[54] See the NGSIM web site: www.ngsim.fhwa.dot.gov.
[55] See creativecommons.org.

their profits through support and maintenance of the tool." The application is expected to run on the Linux, Windows 2000/XP, and Mac OS platforms. Phase 2 will add multiple data import features, including CAD and graphics files as templates or data. At the time of this writing (May 2006) development is focusing on simulation model enhancements.

University of Washington Urban Simulation and Modeling Project

UrbanSim is an open-source project sponsored and developed by the Center for Urban Simulation and Policy Analysis at the University of Washington, Seattle[56]. UrbanSim is a simulation tool that models economic relationships between actors such as households, businesses, developers, and government policies. It uses the following inputs: population estimates, transportation system plans, employment, economic forecasts, land-use plans, and land-development policies such as density constraints, environmental constraints, and development impact fees. It provides the following future outputs:

- Population distributions
- Households by type (e.g. income, age of head, size, number children, housing type)
- Businesses by type (e.g. industry and number of employees)
- Land use by type (user-specified)
- Units of housing by type
- Square footage of nonresidential space by type
- Densities of development by type of land use
- Prices of land and improvements by land use

Funding for UrbanSim has come from the National Science Foundation, the Puget Sound Regional Council, IBM, Google, King County, the Oregon Department of Transportation, the Wasatch Front Regional Council, and the Houston-Galveston Council of Governments, and FHWA. Technical support is provided through the UrbanSim Commons wiki, which provides a means for users to exchange information and resources[57]. Some current UrbanSim users include: Paris, France; Amherst, Massachusetts; Northwing of the Randstad in the Netherlands; Tel Aviv, Israel; and the Wasatch Front in Utah.

UrbanSim uses other open-source packages, including Python, MySQL, and GDAL[58]. It has been tested with Windows XP, SUSE Linux, and Mac OS X (PPC and Intel). UrbanSim uses the GPL open-source license.

[56] For UrbanSim, see cuspa.washington.edu.
[57] For UrbanSim Commons, see urbansimcommons.org.
[58] For GDAL, the Geospatial Data Abstraction Library, see www remotesensing.org/gdal.

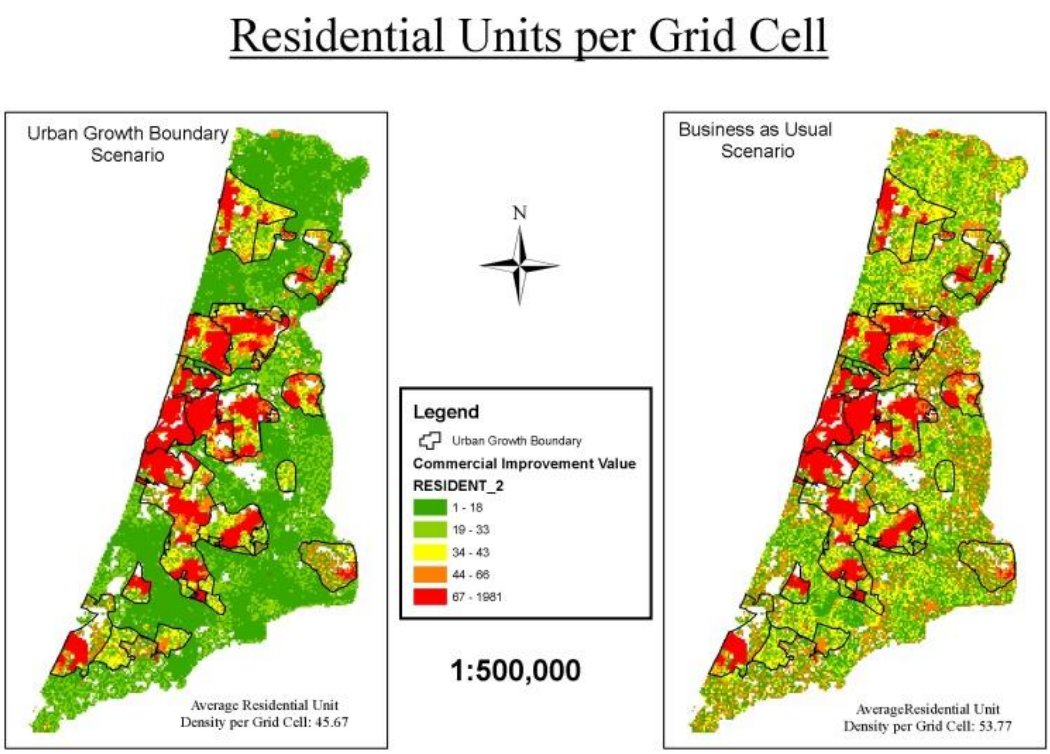

Figure 26: UrbanSim open-source modeling analysis for Tel Aviv metropolitan area [51]

TRB IDEA Program Dynamic Timetable Generator

The Dynamic Timetable Generator project is a prototype proof-of-concept open-source project from the IDEA (Innovations Deserving Exploratory Analysis) program. The IDEA program is sponsored by the Transportation Research Board (TRB) and the National Academy of Science (NAS). The IDEA program funds promising research that uses unproven innovations for highways, rail, safety, and transit [100]. Projects are selected based on innovation, benefits, and science. Transit IDEA projects focus on transit security, bus transit, and innovations that increase ridership.

The Dynamic Timetable Generator project goal is to provide a general-purpose open-source tool that dynamically generates transit timetables for customers accessing a transit web site. Transit agencies benefit from timely and accurate web site updates. A simplified architecture diagram is shown in Figure 27. Project participants include the Oregon Tri-County Metropolitan Transportation District (TriMet)[59], King County Metro[60] in Seattle, Washington, the Chicago Regional Transit Authority,[61] and the New York State DOT Passenger Transport Division[62] [43,100]. The project emphasizes the use of public standards because of anticipated use by a wide range of agencies using diverse operating systems. Standards used include XML

[59] For TriMet, see www.trimet.org.
[60] For King Country Metro, see transit metrokc.gov.
[61] For the Chicago Regional Transit Authority, see www.rtachicago.com.
[62] For the New York State DOT, see www.nysdot.gov.

(eXtensible Markup Language), XSLT (eXtensible Stylesheet Language and Transformation), and the Transit Communications Interface Profile (TCIP). Developed software code will be licensed using an open-source license.

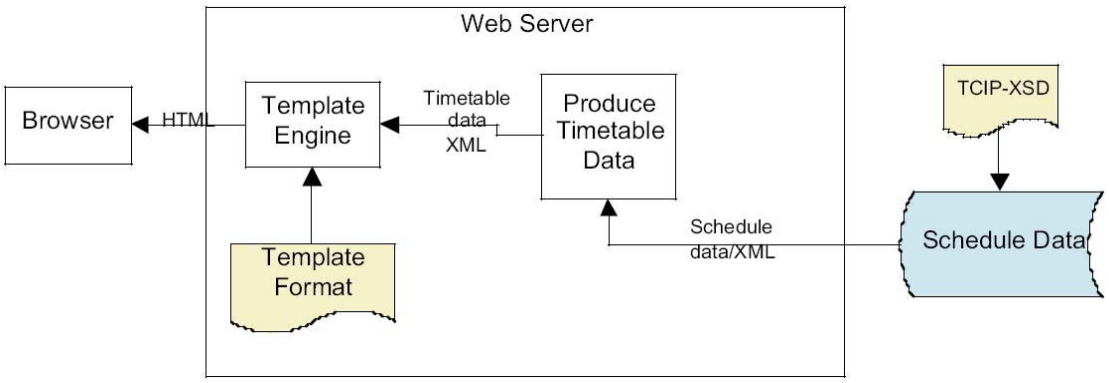

Figure 27: Prototype Dynamic Timetable Generator Architecture [100]

Oregon Tri-County Metropolitan Transportation District TimeTable Publisher

The TimeTable Publisher[63] is an open-source project of the Oregon Tri-County Metropolitan Transportation District (TriMet).[64] The goal of TimeTable Publisher is to provide a general-purpose open-source tool that dynamically generates transit timetables for customers accessing a transit web site. TriMet was a participant in TRB's prototype open-source Dynamic Timetable Generator project (see pg. 57). TimeTable Publisher is based on some ideas from TRB's prototype IDEA project and has benefited from a number of lessons learned during that prototype development. TriMet will release TimeTable Publisher using an open-source license. TriMet uses OSS extensively and is interested in collaborating with other transit agencies, including the possibility of creating an open-source transit foundation.[65] TimeTable Publisher uses the Apache HTTP Server, Linux, and Tomcat, among others. TriMet is also working with Google Inc. on the Google Transit project,[66] which uses the Creative Commons Attribution-ShareAlike license.[67] TriMet reports the following organizational benefits for using OSS [101]:

- Shared costs, risks, and lessons learned
- Software support and maintenance can be extended indefinitely
- Approach is proven and in use
- Control over technical destiny
- Development and service base larger under OSS
- Benefits smaller agencies
- Consistent user interfaces

[63] For TimeTable Publisher, see timetablepublisher.com.
[64] For TriMet, see www.trimet.org.
[65] For a discussion of use of OSS in Transit and the use of software foundations, see P. Okunieff, "Transit and Open Source: Is It an Option?" www.consystec.com/docs/IDEA_TOSSI_Final_062905_V2.1.pdf
[66] For Google Transit, see maps.google.com/transit and for the open-source Google Transit Feed Specification, see code.google.com/transit/spec/transit_feed_specification.htm.
[67] For the Creative Commons ShareAlike License, see creativecommons.org/licenses.

TriMet reports the following technical benefits for using OSS:

- Ability to customize code
- Flexibility
- Modularity
- Shorter development cycles
- Re-usable code
- Ability to scale application to meet specific performance criteria
- More technical support and resources due to broader base
- More Assessable: ADA, General, Documentation
- Fosters innovation and competition

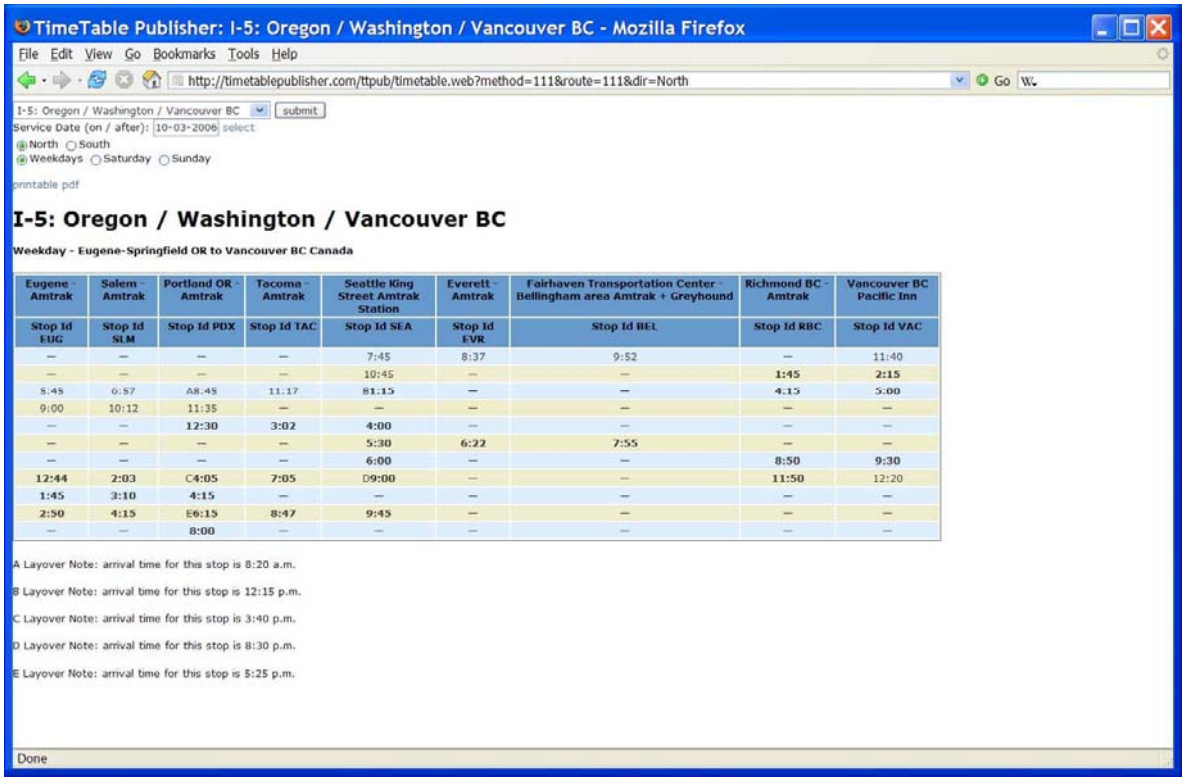

Figure 28: TriMet's Transit TimeTable Publisher [101]

SECTION 10:
CASE STUDIES OF ITS HARDWARE PROJECTS USING OPEN-SOURCE SOFTWARE

This section discusses case studies of ITS hardware projects that *use* OSS. The prior sections discussed projects that are more software-oriented, while this section and the next focus on explicit hardware systems. The focus is on ATMS related projects; however, other transportation related projects are discussed. Each case study strives to answer who, why, what, and how, with respect to OSS. A significant point is that the vast majority of the case studies in this report used commodity x86 and x64 hardware in addition to OSS.

Peek Traffic Inc. Linux-Based Commercial Traffic Data Recorder

Peek Traffic Inc. sells a Linux-based automatic traffic data recorder, the ADR-6000 [24]. It is a rack-mount unit with a Pentium-class processor.[68] The Texas Transportation Institute performed an evaluation of the ADR-6000 and three similar units [44]. The researchers noted that the unit was comparable to the others and fell near the bottom of the price range.

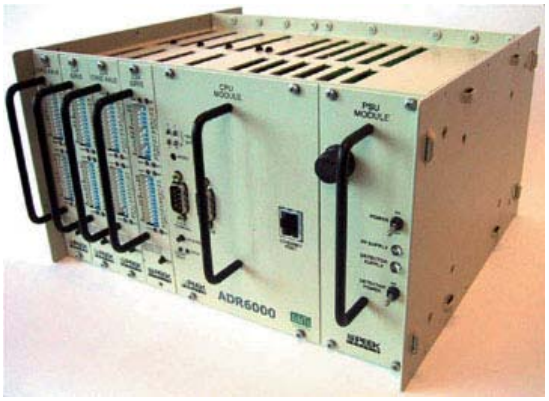

Figure 29: Peak ADR-6000 Linux based automatic data recorder [24]

City of Valencia, Spain Video Streaming System for Urban Traffic Control

The city of Valencia Spain recently deployed a traffic management system that uses OSS to stream video over TCP/IP [49]. The city has 600 traffic cameras with plans to increase that to 1000. The guiding philosophy used to build the system was to use COTS and OSS components that follow open standards, and avoid proprietary systems. TCP/IP video cameras were used instead of traditional CCTV to provide increased scalability, lower cabling costs, and facilitate providing video streams to the public over the Internet. The video cameras use the MPEG-4 codec. The system uses Linux, the OSS database MySQL, the OSS video player VLC, and the OSS multimedia system FFMPEG[69], among others [83]. Figure 30 provides a diagram of the main system components.

[68] For ADR-6000 specifications, see www.ustraffic net/products/data/ADR-6000-05.pdf.
[69] See www mysql.com, www.videolan.org, and ffmpeg mplayerhq hu.

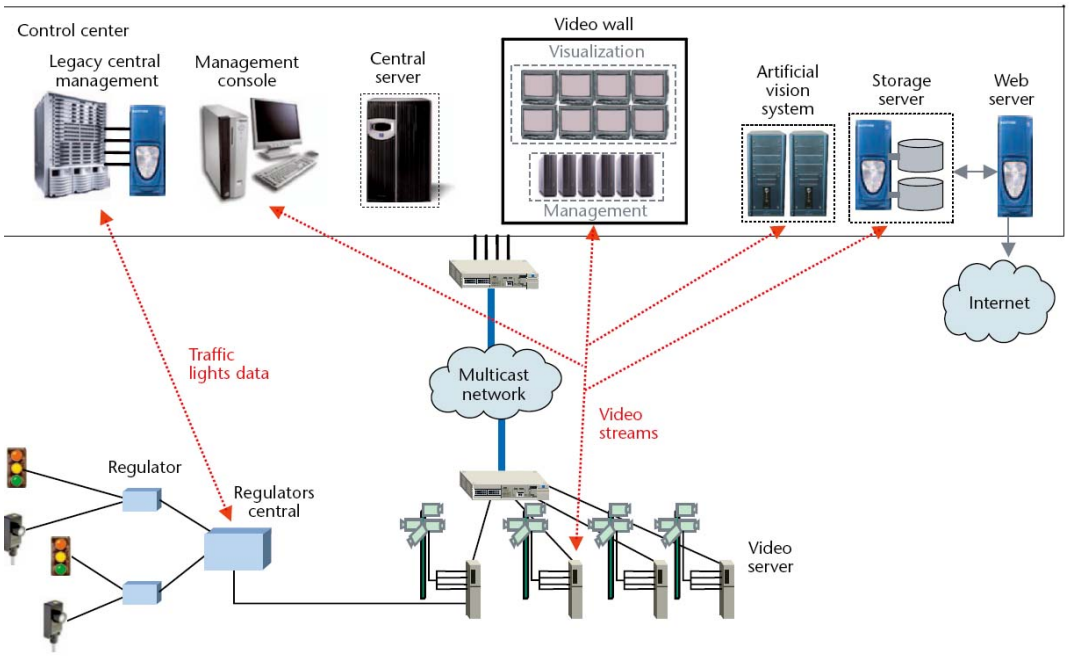

Figure 30: System components of TMC, Valencia, Spain [49]

SECTION 11:
CASE STUDIES OF ITS OPEN-SOURCE HARDWARE PROJECTS

This section discusses case studies of ITS hardware projects that *are* open-source projects—they attempt to build a community of users sharing software code, experience, improvements, etc. The community benefits from these contributions. Projects in this category use some type of OSS license to grant users access to project artifacts. See Appendix D for information on open-source licenses (pg. 89). Prior sections discussed ITS projects that are more software-oriented, while Section 10 presented *hardware projects* that *use* open source, but are not actually open-source projects. The focus is on ATMS related projects; however, other transportation related projects are discussed. Each case study strives to answer who, why, what, and how, with respect to OSS. A significant point is that the vast majority of the case studies in this report used commodity x86 and x64 hardware in addition to OSS.

U.S. DOT Advanced Transportation Controller

The Advanced Transportation Controller (ATC) project *uses* OSS and *is* an OSS project. It is a standardization effort initiated by the U.S. Department of Transportation, and sponsored by AASHTO, ITE, and NEMA (see also ATMS Trends, pg. 37). It has been recognized as one of the most important standards for the ITS community. ATC will provide an open software and hardware platform for a wide variety of ITS applications, and is viewed as similar to a ruggedized field-deployed personal computer. Table 10 shows some of the anticipated applications. The draft standard is near completion and adoption. The standard stipulates that the "ATC shall use a Linux operating system (O/S) and shall include standard POSIX libraries for application support including real-time extensions of POSIX 1003" [11]. Figure 31 (pg. 64) shows the layered structure of ATC. The standardized API layer insulates applications from changes to the operating system and hardware.

Table 13: Anticipated ATC applications [11]

Traffic Signal Highway	Rail Intersections
Traffic Surveillance	Speed Monitoring
Lane Use Signals	Incident Management
Communications	Highway Advisory Radio
Field Masters	Freeway Lane Control
Ramp Meter	High Occupancy Vehicle Systems
Variable/Dynamic Message Signs	Access Control
General ITS beacons	Roadway Weather Information Systems
CCTV Cameras	Irrigation Control

The ATC standardization effort is needed for several reasons. First, there is an increasing need to integrate controllers that were previously incompatible (e.g. the 2070 and NEMA controllers), both for functional and standardization reasons. Second, increasingly sophisticated ITS applications require supporting a broader set of applications, faster communication, and the ability to evolve over time. The ATC standard identifies this need as a broad ITS trend: "as the current trend continues towards distributing more of the intelligence of ITS out closer to the field, there is an increasing demand for more and more capable field deployable devices. This hardware must run more sophisticated applications software and operate in modern networking environments. The ATC Controller is intended to address these needs" [11]. It is interesting to

note that the ATC standardization effort started with an attempt simply to standardize the API layer and not the operating system. Clearly at some point the standardization committee came to the conclusion that an open-source operating system (specifically, Linux) could be adopted into the standard without sacrificing portability, future expansion, reliability, hardware options, or a healthy ATC hardware and software market.

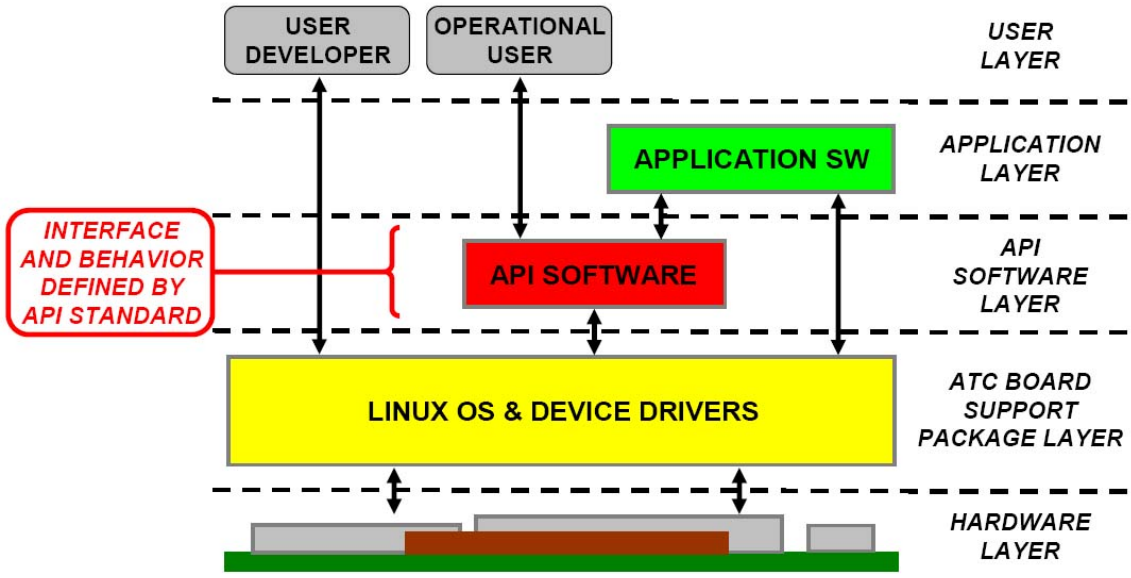

Figure 31: ATC software layered organization [10]

European ERTICO Global System for Telematics Open System

The European Global System for Telematics Open System project both *uses* OSS and *is* an OSS project. ERTICO is the European equivalent to ITS America, and sets standards, funds research, and coordinates projects. The Global System for Telematics (GST) is an ongoing project with the goal of facilitating the creation of an open market for in-vehicle ITS services [15]. The GST Open Systems sub-project is delivering an open telematics framework, consisting of the specifications, architecture, and a reference implementation for in-vehicle applications. The open telematics framework will be used by service providers, car manufacturers, and mobile end-users. In-vehicle applications will be distributed by service providers from control centers. The GST Open Systems project provides specifications and example implementations of these systems for service providers and car manufacturers implementing these systems.

The Open Systems project uses open standards and open-source. Mobile applications are deployed from control centers via web applications and servers using the open-source GPL license. Applications run on the open-source JBoss[70] J2EE application server. The SyncML[71] synchronization standard is used, which is based on the Sync4J 1.4.8 open-source implementation. Other open standards used include HTTP, SOAP, and TCP/IP (see Figure 32 for

[70] See www.jboss.org.
[71] See www.openmobilealliance.org.

more). The open-source Eclipse[72] development environment is also used. Both Linux and Windows XP are used for development.

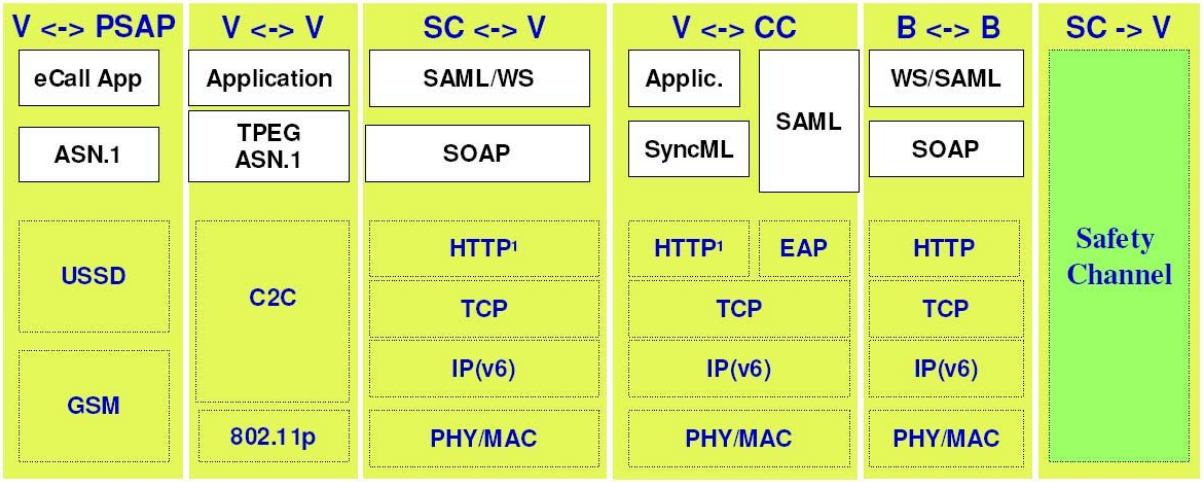

Figure 32: ERTICO GST Open System protocol stack [108]

[72] See www.eclipse.org.

SECTION 12:
CONCLUSIONS

Summary

This report has three objectives:

- to summarize the history and current developments in ATMS software and hardware,
- to summarize the history, developments, strengths, and weaknesses of OSS, and
- to summarize case studies of OSS and commodity hardware use in ATMS projects in the reference literature.

Section 2 discusses how and why technology markets change, the nature of innovations, product lifecycles, disruptive innovation, and commoditization of the computer hardware market. Section 3 presents OSS history and recent developments. Sections 4-6 discuss how OSS is different, software engineering, types of software development, and OSS strengths and weaknesses. Section 7 summarizes ATMS history, goals, and trends. Sections 8 and 9 overview case studies of OSS use and OSS projects in ATMS and ITS software, while Sections 10 and 11 summarize case studies of OSS use and OSS projects in ATMS and ITS hardware. Finally, the appendices discuss lock-in, popular OSS projects, relevant ATMS standards, OSS licenses, and semantic modeling.

Conclusion

The ATMS and ITS markets are being driven both internally by customer needs and externally by advances in technology (see Figure 34, pg. 70). Improvements in semiconductor process technology continue to push commoditization of sensors, processors, communication, and decentralization of processing intelligence, providing the capability for real-time decision-making. ATMS customer needs are also driving change. As ITS build-out continues into lower-density population areas, cost-sensitivity increases. The primary obstacle to implementing ATMS is no longer technology—rather it is cost. Homeland Security requirements appear to be a long-term trend driving some ATMS functional requirements such as video surveillance and increased IT security. Finally, the ongoing need to reduce congestion, lower accident rates, and increase efficiency is driving needs for further innovation, ATMS interoperability, more and better real-time data, and the ability to make traffic management decisions in real time. As recognized by the National ATC standardization effort, these trends are driving the need for more sophisticated field devices. This, combined with lower hardware prices and the price sensitivity of further ATMS build-out are driving the need for software that is less expensive and more secure, reliable, standards-based, and innovative.

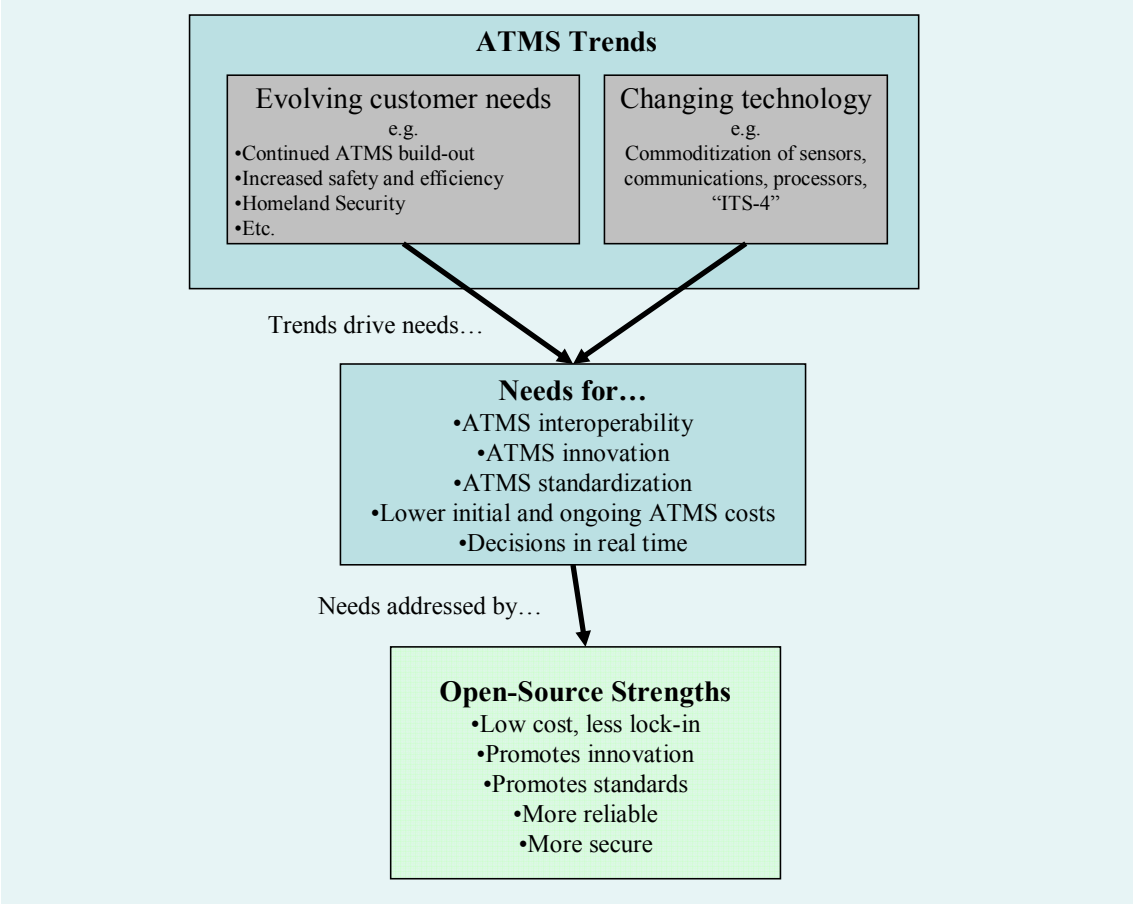

Figure 33: ATMS trends, needs, solutions (simplified)

Improvements in safety, congestion, and efficiency have already been achieved with ITS applications and proprietary software. This report details the use of OSS to build ITS and ATMS applications. Each of the ITS projects covered in the case studies used OSS in one of two ways:

1. The first type of project case study used open-source software to implement a closed-source application: Ten of the case studies used this approach. Project organization is more or less unchanged compared to projects using proprietary products.

2. The second type of project used open-source software to implement an open-source application. Five of the case studies used this approach. These projects are organized to share project source code, datasets, test results, etc. with a community of users who may contribute to the project in some form. The community as a whole benefits from these contributions. Projects in this category use some type of OSS license to give a community ongoing access to project artifacts.

Benefits provided by OSS and its unique development model must be balanced with a consideration of concerns, including lack of trained staff, less user-friendliness, and inconsistent quality. An understanding of the OSS development model is crucial for setting expectations. For example, benefits gained from peer review are proportional to the number of individuals

involved. As shown in the case studies, measured use of OSS with a consideration of OSS strengths and concerns can provide immense benefits to an organization.

There are three conclusions reached here. First, a number of ATMS and ITS projects have constructed applications using open-source software. For example, Minnesota's IRIS ATMS system, Oklahoma's Statewide distributed ITS, and the National ATC standard for field devices all use OSS. In addition, others have used OSS to implement large and complex non-ITS applications, including the FAA's technical refresh of the Nation's Real-time Enhanced Traffic Management System, and a number of corporations that run their businesses on OSS and commodity hardware, such as Google, E*TRADE, Sabre, Travelocity, Yahoo!, and Amazon.

Second, as summarized in Figure 34, some of the benefits reported in the case studies related to the OSS development model parallel trends in ATMS and ITS. For example, high deployment costs are a concern for continued ATMS build-out in California. The need for increasingly sophisticated software applications in field devices and TMCs are also driving higher software costs. These cost concerns parallel OSS cost benefits reported in some of the case studies (e.g. see the Mn/DOT IRIS System, pg. 43). Reduced lock-in is closely associated with zero or low software licensing costs. Needs for higher ATMS reliability and security parallel reliability and security strengths reported with the OSS model, and may benefit ATMS applications.

Third, in the ITS field in general, there appears to be increasing interest in collaboration. This parallels the general requirement that ATMS and ITS systems increase functional interoperability. One third of the cited case studies (5) are themselves open-source projects using open-source licenses to ensure community access to artifacts. A number of individuals contacted for this report and involved in the case studies expressed interest in sharing their proprietary work with others in the form of open-source ATMS projects. The open-source development model facilitates collaboration, the exchange of ideas, software code, and pooled funding which spreads development risks. DOTs, cities, and counties may benefit from continuing this collaborative approach. Collaborative access to source code may also increase competition among prospective firms because of lowered entry barriers. Firms or individuals that did enhance an OSS ATMS could not claim a proprietary right to the source code they wrote, potentially reducing lock-in for State DOTs. However, adopted enhancements would gain them visibility and recognition, which would increase their ability to win contracts for future enhancements.

These conclusions taken together indicate strong potential for application of OSS for ATMS in California and the Nation. The current project will continue to investigate California's evolving ATMS requirements, and map these against the capabilities and potential for application of OSS, in order to test and demonstrate the applicability and benefits relative to California's needs.

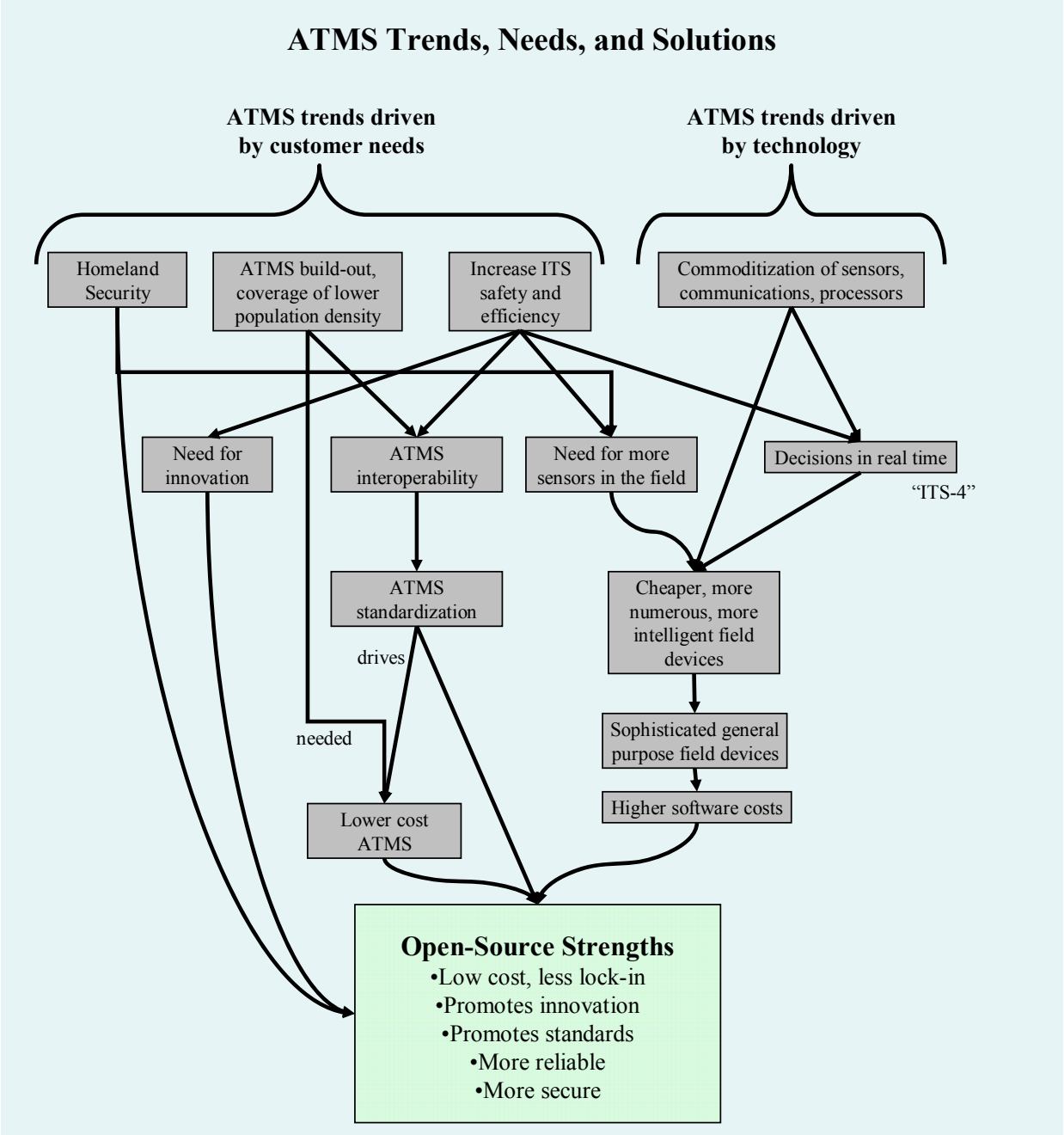

Figure 34: ATMS trends, needs, and solutions (detailed)

REFERENCES

1. "Size of the U.S. Computer Software Industry," Office of Technology and Electronic Commerce, United States Department of Commerce, 1997.
2. "Software Errors Cost U.S. Economy $59.5 Billion Annually," National Institute of Standards and Technology, http://www.nist.gov/public_affairs/releases/n02-10.htm, 2002.
3. "Enhancing the Usability of an Intersection Collision Avoidance Simulation Model," U.S. Department of Transportation, http://www.volpe.dot.gov/sbir/sol03/sec8full.html, 2003.
4. "The CHAOS Report," The Standish Group, http://www.standishgroup.com, 2004.
5. "ITS Goals Supported by Market Packages, National ITS Architecture," U.S. Department of Transportation, http://www.iteris.com/itsarch/html/mp/itsgoalsbymp.htm, 2005.
6. "ITS Goals, National ITS Architecture," U.S. Department of Transportation, http://www.iteris.com/itsarch/html/mp/itsgoals.htm, 2005.
7. "Linux Outlook," InformationWeek Research Brief, http://i.cmpnet.com/infoweek/1057/IWKLinuxOutlook-2005.pdf, 2005.
8. "National ITS Architecture V5.1," U.S. Department of Transportation, http://www.iteris.com/itsarch/, 2005.
9. "1969 Chevrolet Astro III Concept Car," Fotos de Carros Web Site, http://www.fotosdecarros.com, 2006.
10. "Advanced Transportation Controller (ATC) Application Programming Interface (API) Software Requirements Specification (SRS)," Institute of Transportation Engineers, http://www.ite.org/standards/atc/APISRS0204.pdf, 2006.
11. "Advanced Transportation Controller (ATC) Standard, Version 5.2a," Institute for Transportation Engineers, http://www.ite.org/standards/atc, 2006.
12. "Common ARTS History," Federal Aviation Administration, http://www.faa.gov/ats/atb/Sectors/Automation/CommonArts/history.htm, 2006.
13. "Denver ARTCC, Traffic Management System Assures the Maximum Safety and Efficiency," http://www.nw.faa.gov/ats/zdvartcc/tmu_tools.html#110, 2006.
14. "Encyclopaedia Britannica Website," http://britannica.com, 2006.
15. "ERTICO GST Project Web Page," http://www.ertico.com/en/activities/projects_and_fora/gst.htm, 2006.
16. "Federal Aviation Administration FAA Saves $15 Million by Migrating to Red Hat Enterprise Linux Desktop," Red Hat Inc., http://www.redhat.com/rhel/informationcenter/successstories/government/faa.html, 2006.
17. "IT Strategic Headquarters Web Site," Prime minister of Japan and His Cabinet, http://www.kantei.go.jp/foreign/policy/it/index_e.html, 2006.
18. "ITS Research Laboratory," University of Oklahoma, http://its.ou.edu, 2006.
19. "ITS Standards Program Website," U.S. Department of Transportation, http://www.standards.its.dot.gov/default.asp, 2006.
20. "Managing Rural Roads in Local Agencies and on Indian Reservations, Rural ITS - Real World Solutions," Department of Transportation ITS web site, http://www.its.dot.gov/rural/CaseDetails.asp?ID=20, 2006.
21. "Minnesota Department of Transportation, RTMC Homepage," http://www.dot.state.mn.us/tmc/index.html, 2006.
22. "Next Generation Simulation Program Web Page," http://ngsim.fhwa.dot.gov/, 2006.

23. "Oasis Web Site," OASIS, http://www.oasis-open.org/news/oasis-news-2006-05-08.php, 2006.
24. "Peek Traffic Inc. Web Page," http://www.ustraffic.net, 2006.
25. "Red Hat Technical Support Call Handling," Red Hat, https://www.redhat.com/support/service/guide/, 2006.
26. "Safety Dynamics Inc.," http://www.safetydynamics.net/products.html, 2006.
27. "Secunia Inc. Security Reports," http://secunia.com/product, 2006.
28. "Transportation Futuristics Virtual Museum," University of California Berkeley, http://www.lib.berkeley.edu/news_events/exhibits/futuristics/index.html, 2006.
29. V.R.B. B. Boehm, "Software Defect Reduction Top 10 List," *IEEE Computer Society*, **34**(1): pp. 135-137, 2001.
30. C. Bahrmann, Pennsylvania State University, Department of Meteorology, http://www.met.psu.edu/, 2006.
31. M.E. Ben-Akiva, "Calibration and Evaluation of MITSIMLab in Stockholm," http://web.mit.edu/its/papers/CALIBR2.PDF, 2002.
32. T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American,* 2001.
33. T. Bollinger, "Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defense," Mitre Corporation Rept. # MP 02 W0000101, 2003.
34. S. Bratt, "Weaving a Web for the Next Generation of Science," W3C Presentation, http://www.w3.org/2004/Talks/1109-sb-gsaWebSci/Overview.html, 2004.
35. T. Brusehaver, "Linux in Air Traffic Control," http://www.linuxjournal.com/article/7066, 2004.
36. W. Burghout, "ITS Features in MITSIMLab," Centre for Traffic Engineering and Simulation, Department of Infrastructure and Planning, Royal Institute of Technology, http://www.infra.kth.se/ctr/publikationer/ctr2000_04.pdf, 2000.
37. B. Cantrill, "The Economics of Software," http://blogs.sun.com/bmc/20040828, 2004.
38. B. Chelf, "Measuring Software Quality a Study of Open Source Software," Coverity Inc., http://www.coverity.com, 2006.
39. C.M. Christensen, *The Innovator's Dilemma*, Harper Business, 2000.
40. G. Clarke, "MySQL Destined for 'Majority' Market Share," The Register, http://www.theregister.co.uk/2005/10/18/mysql_marketshare_numbers/, 2005.
41. G. Clarke, "Open Source Taking over Europe," The Register, http://www.theregister.co.uk/2005/10/21/opensource_government/, 2005.
42. ComputerWorld, "Sony Online Opts for Open-Source Database over Oracle," http://www.computerworld.com/databasetopics/data/software/story/0,10801,109722,00.html, 2006.
43. ConSysTec Inc., "Transit Projects Summary Web Page," http://www.consystec.com/transproj.html, 2006.
44. R.P. D. Middleton, "Evaluation of Vehicle Detection Systems," Texas A&M University, College Station, Texas 77843-3135 Rept. # 2119-1, 2002.
45. Delcan Inc., "RIITS Project Outreach and Marketing Package--Final," RIITS, Los Angeles County Metropolitan Transportation Authority, http://www.riits.net/pdf/Final_2.6_051904-RIITS.pdf, 2004.
46. Delcan Inc., Personal Communication, 2006.

47. K. Edwards, "Epistemic Communities, Situated Learning and Open Source Software Development," http://opensource.mit.edu/papers/kasperedwards-ec.pdf, 2001.
48. J. Ellison, University of Maryland, CATT Laboratory, http://www.cattlab.umd.edu/, 2006.
49. M. Esteve, C.E. Palau, and T. Catarci, "A Flexible Video Streaming System for Urban Traffic Control," *Multimedia IEEE*, **13**(1): pp. 78-83, 2006.
50. J. Evers, "Developers Fast to Fix Open Source Bugs," ZDNet News, http://news.zdnet.com/2100-1009_22-6057669.html, 2006.
51. D. Felsenstein, "UrbanSim Application for the Tel Aviv Metropolitan Area," Department of Geography & Institute for Urban and Regional Studies Hebrew University of Jerusalem, http://urbansimcommons.org/IsraelProject, 2006.
52. Free Software Foundation, "GNU General Public License Version 2," http://www.gnu.org/copyleft/gpl.html, 1991.
53. J. Gapper, "A Threat to the Fragile Linux Ecosystem," http://news.ft.com/cms/s/7beb0326-d2ed-11da-828e-0000779e2340.html, 2006.
54. D. Gibson, Conversation with, 2006.
55. J. Giles, "Internet Encyclopaedias Go Head to Head," *Nature*, 2005.
56. J. Havlicek, Conversation with, 2006.
57. R.C. Huck, "A Low-Cost Distributed Control Architecture for Intelligent Transportation Systems Deployment in the State of Oklahoma," http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1520173, 2005.
58. R.C. Huck, Conversation with, 2006.
59. ITS America, "ITS America 10 Year Plan," ITS America, http://www.itsa.org/itsa/files/pdf/National10YearPlanITSExecSummary.pdf, 2002.
60. C.M. Johnson, "The National ITS Program: Where We've Been and Where We're Going," *Public Roads On-line*, Public Roads On-line, pp, 1997.
61. R.W. Joode, *Protecting the Virtual Commons, Self-Organizing Open Source and Free Software Communities and Innovative Intellectual Property Regimes*, 2003.
62. K. Kenedy, "Ballmer: Microsoft to Focus on Linux Competition, Software as a Service, Internet Advertising," *Information Week*, 2006.
63. C.A. Kenwood, "A Business Case Study of Open Source Software," Mitre Corp., http://www.mitre.org/work/tech_papers/tech_papers_01/kenwood_software/kenwood_software.pdf, 2001.
64. J.M. Kranig, Director of Traffic Operations for the Regional Transportation Management Center, Office of Traffic, Security and Operations, Minnesota Department of Transportation, 2006.
65. S. Lacy, "Oracle's Open-Source Shopping Spree," Business Week, http://www.businessweek.com/technology/content/feb2006/tc20060209_810527.htm, 2006.
66. M. LaMonica, "Oracle Buys Open Source Database Firm," CNet News, http://news.com.com/Oracle+buys+open source+database+firm/2100-7344_3-5892632.html, 2005.
67. M. LaMonica, "Microsoft to 'Open the Doors' of Linux Labs," CNet News, http://news.com.com/Microsoft+to+open+the+doors+of+Linux+labs/2100-7252_3-6058196.html?tag=st.rn, 2006.

68. M. Lamonica, "States Struggling to Deal with Digital Documents," CNet News, http://news.com.com/States+struggling+to+deal+with+digital+documents/2100-1014_3-6064793.html?tag=nl, 2006.
69. Library of Congress, "Copyright Office Website," United States Copyright Office, http://www.copyright.gov/, 2006.
70. Los Angeles County Metropolitan Transportation Authority, "Task 3.5, Software Design Document Version 1.0," Regional Integration of Intelligent Transportation Systems Project, http://www.riits.net/pdf/SDD_v10.pdf, 2003.
71. Los Angeles County Metropolitan Transportation Authority, "Integration Instructions, Version 1.0," Regional Integration of Intelligent Transportation Systems Project, http://www.riits.net/pdf/Integration%20Instructions%20V2.pdf, 2004.
72. Los Angeles County Metropolitan Transportation Authority, "RIITS Guidebook," Regional Integration of Intelligent Transportation Systems Project, http://www.riits.net/pdf/RIITS-Guidebook-Final.pdf, 2004.
73. F. Marinescu, "Advanced Message Queue Protocol to Commoditize Messaging," InfoQueue, http://www.infoq.com/news/amq, 2006.
74. Massachusetts Institute of Technology, "MITSIMLab Brochure," http://web.mit.edu/its/papers/MITSIM2001.pdf, 2006.
75. Massachusetts Institute of Technology, "MITSIMLab Web Page," http://web.mit.edu/its/mitsimlab.html, 2006.
76. K. McIsaac, "The Future of Software," MetaGroup, http://www.oracle.com/global/in/corporate/presentations/owmumbai/Tech_Keynote-The_Future_of_Technology-Kevin_McIssac-Meta_Group.pdf, 2004.
77. G. Moody, *Rebel Code*, 2002.
78. G. Moody, "Does Dual Licensing Threaten Free Software?," Linux Journal, http://linuxjournal.com/node/1000069, 2006.
79. R. Naraine, "DHS Funds Open-Source Security Project," eWeek, http://www.eweek.com/article2/0,1895,1909946,00.asp, 2006.
80. Netcraft, "Netcraft Usage Statistics," http://www.netcraft.com, 2006.
81. Open Source Initiative, "History of the OSI," http://www.opensource.org/docs/history.php, 2006.
82. M. Pack, "Presentation of Regional Integrated Transportation Information System (Ritis)," University of Maryland, CATT Laboratory, http://www.cattlab.umd.edu/media/other/RITIS for Joe Geckle.ppt, 2006.
83. C.E. Palau, Conversation with, 2006.
84. J.F. Paniati, "SAFETEA-LU Implementation ITS & Operations," ITS America 2006 Annual Meeting and Exposition, http://ops.fhwa.dot.gov/speeches/its_america/2006/paniati_50706/index.htm, 2006.
85. B. Park, "Developing Efficient Data Archive Designs for the State of Virginia," TRB Publication, http://trb.mtc.ca.gov/urban/adus/Virgina%20ADUS.pdf, 2003.
86. B. Park, "Web-Based Congestion Monitoring Map for Nova Smart Traffic Signal System," http://cts.virginia.edu/docs/UVACTS-13-0-83.pdf, 2003.
87. B. Perens, "The Emerging Economic Paradigm of Open Source," Cyber Security Policy Research Institute, George Washington University, http://perens.com/Articles/Economic.html, 2005.
88. R.S. Pressman, *Software Engineering*, McGraw-Hill, 2001.

89. E.S. Raymond, *The Cathedral and the Bazaar*, 2001.
90. R. Rhodes, "Open Source and Linux," IBM, http://www-1.ibm.com/linux/industry/opensource.shtml, 2005.
91. P.H. Salus, *A Quarter Century of Unix*, 1994.
92. P.H. Salus, "Unix at 25," http://wolfram.schneider.org/bsd/ftp/article/rt3.htm, 1997.
93. L. Saxton, "Mobility 2000 and the Roots of IVHS," *NHS Review*, pp. 11-26, 1993.
94. D. Smallen, "How Transportation Systems Talk to Each Other--Standards for Intelligent Transportation Systems," *Public Roads*, (Sept-Oct), 1999.
95. STL, "ADMS Virginia " Smart Travel Lab, University of Virginia, http://cts.virginia.edu/stl_adms.htm, 2006.
96. J.M. Sussman, "Perspectives on Intelligent Transportation Systems," 2005.
97. M.H. T. Dingus, S. Jahns, "Development of Human Factors Guidelines for Advanced Traveler Information Systems and Commercial Vehicle Operations: Literature Review," Federal Highway Administration Rept. # FHWA-RD-95-153, 1996.
98. J.B. Taylor, *Principles of Macroeconomics*, Houghton Mifflin, 1998.
99. Transportation Research Board, "An Accident Analysis System for Traffic Engineering Decision Support on a Statewide Basis—SAFE-T," http://rip.trb.org/browse/dproject.asp?n=9119, 2003.
100. Transportation Research Board, "New Ideas for Transit, Transit Idea Program Annual Progress Report," http://onlinepubs.trb.org/onlinepubs/sp/transit-idea_report_jan2005.pdf, 2005.
101. Tri-County Metropolitan Transportation District, "Transit Oss and Data Sharing Meeting, Oss Benefits for Transit," Portland, pp. 1, 2005.
102. Turner-Fairbanks Highway Research Center, "Collaborative Research on Road Weather Observations and Predictions by Universities, State DOTs and National Weather Service Forecast Offices, Developing an Interactive Mesonet for PennDOT," Federal Highway Administration, http://www.tfhrc.gov/its/pubs/04109/, 2004.
103. University of Oklahoma, "Intelligent Transportation Systems Laboratory," http://its.ou.edu, 2006.
104. University of Virginia Center for Transportation Studies, "Smart Travel Lab, Archived Data Management System," http://cts.virginia.edu/stl_adms.htm, 2006.
105. J.C. V. Valloppillil, "Microsoft Halloween Documents," http://www.catb.org/~esr/halloween/, 1998.
106. L. Vaas, "Linux Muscling to the Top in Oracle Shops " http://www.eweek.com/article2/0,1895,1945810,00.asp, 2006.
107. M. Välimäki, "The Rise of Open Source Licensing: A Challenge to the Use of Intellectual Property in the Software Industry," pp, 2005.
108. E. Vermassen, "GST Open Systems White Paper," ERTICO Open Systems, http://www.gstproject.org/os/documents/DOC_GST_OS_DEV_White_Paper.pdf, 2006.
109. J. Viega, "Open Source Security: Still a Myth," O'Reilly Network, http://www.onlamp.com/pub/a/security/2004/09/16/open_source_security_myths.html?page=1, 2004.
110. W3C, "W3C Semantic Web Activity Page," W3C, http://www.w3.org/2001/sw/, 2006.
111. C.E. Wallace, "ITS Florida—Yesterday, Today, and Tomorrow," ITS Florida, http://itsflorida.org/documents/official_doc/Founding_of_ITSA_FL.doc, 2005.

112. C.E. Wallace, "A Series of E-Mail Messages About the Founding of What We Now Call ITS," http://itsflorida.org/documents/official_doc/April-Email_Histories.doc, 2005.
113. L.C. Ware, "Open Source Gains Momentum," CIO Magazine, http://www2.cio.com/research/surveyreport.cfm?id=48, 2002.
114. R. Watson, "Workshop on Application of Remote Sensing Technologies for Disaster Response," 2003.
115. Wikipedia, "Wikipedia Statistics," http://en.wikipedia.org/wiki/Special:Statistics, 2006.

APPENDIX A:
LOCK-IN: WHY DOES A DATABASE COST MORE THAN AN OPERATING SYSTEM?

Some databases cost tens of thousands of dollars. Why? A database is comparatively no more complex than a word processor, spreadsheet, or microprocessor. The simple answer is *lock-in*. That is, after a database-dependent application is written, customers are willing to pay very high prices to keep their applications running. The economic term for this situation is *demand inelasticity* [37]. Other products have high demand inelasticity, e.g. essential medical drugs, and cigarettes. Figure 35 shows a supply/demand curve for this situation. As the quantity available of a certain product decreases, a customer's willingness to pay higher prices increases until the customer reaches the breaking point and rewrites their application using another database. The production of software is unique, in that there is hardly any incremental cost to vendors for providing a nearly unlimited supply of the product. Compare this lock-in situation with a healthy competitive market supply/demand curve in Figure 36. In a healthy market with multiple suppliers and lower demand inelasticity (lower slope on the demand curve), customers can more freely move between different suppliers, which discourages suppliers from raising prices. The high demand inelasticity of software also explains why software vendors are happy to offer limited functionality low-cost or free versions of their products—they are counting on high switching costs ultimately producing more total revenue.

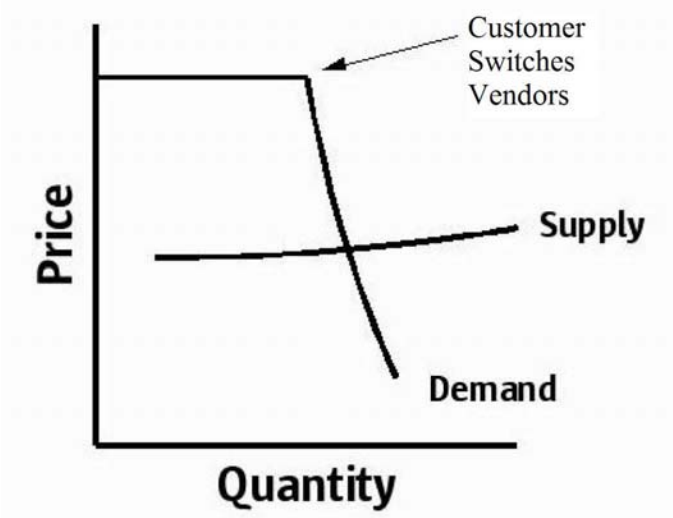

Figure 35: Database demand inelasticity price curve [37]

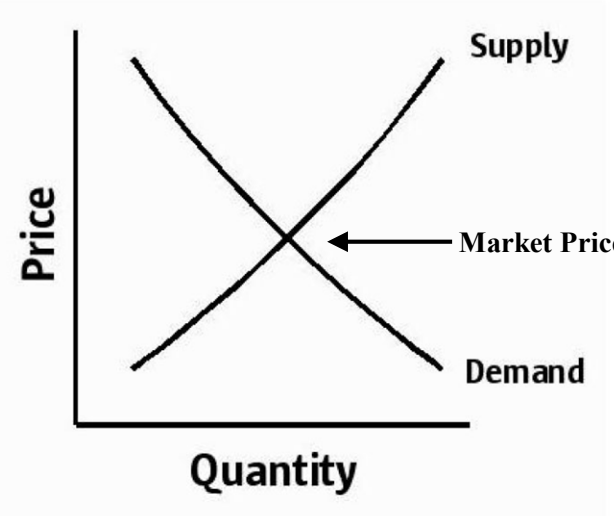

Figure 36: Supply/demand curve for a competitive market [37]

APPENDIX B:
PARTIAL DIRECTORY OF MAINSTREAM OPEN-SOURCE SOFTWARE APPLICATIONS

This appendix provides a partial list of potentially useful mainstream open-source software applications. This is not an all-inclusive list and the intent is simply to orient the reader within the OSS world relative to major OSS projects and project categories. All of the projects listed have a large and active developer and user community, which is an important factor for quality and reliability of an OSS project. All of the non-operating system projects listed have versions that run on Linux and Windows, and many support other operating systems such as BSD, Solaris, Mac OS, and others. The AHMCT Research Center has a long history of using open-source and proprietary software products in a heterogeneous computing environment. The Center has used or is using many of the products below, in research and/or IT operations.

The summary information presented here will provide a general overview of each project; however, for detailed understanding, the reader is referred to the appropriate web site(s) and references.

Apache ActiveMQ Messaging Middleware

Apache ActiveMQ is messaging middleware used by software developers to add messaging capabilities to custom-built applications. It supports clustering, peer networks, discovery, SSL, multicast, and persistence, among other capabilities. It is a full Java Messaging Service 1.1 (JMS) provider. It also integrates into J2EE 1.4 containers. It supports a number of client languages such as Java, C, C++, C#, Ruby, Perl, Python, and PHP. Commercial support is available. ActiveMQ was originally formed and funded by JP Morgan and Chase to eliminate the use of proprietary messaging products and associated licensing fees: "Banks spend a lot of money just to send messages inside their own bank...AMQ should help to commoditize the messaging industry much like web servers have been commoditized by Apache" [73]. ActiveMQ is being used in a production environment by JP Morgan as a global trading system consisting of 800 users across five companies and data centers. "We have implementations from multiple companies running Java, C++, C#, running across Windows, Linux, and Solaris" [73]. ActiveMQ uses the Apache license. For further information, see www.activemq.org.

Apache HTTP Web Server

The Apache HTTP web server is by far the most popular HTTP web server in use (see Figure 8, pg. 9). Apache was initially developed at the National Center for Supercomputing Applications at the University of Illinois at Urbana Champaign. The Apache web server is maintained and developed by the Apache Software Foundation. It uses the Apache license. For further information, see www.apache.org.

BSD Operating System

BSD (Berkeley Software Distribution) is a Unix-like operating system. BSD has forked into a number of parallel projects such as NetBSD[73], FreeBSD[74], and OpenBSD[75], among others. These are derivates of the original AT&T Unix distribution from the early 1970s[76]. The BSD license is used. For further information, see en.wikipedia.org/wiki/BSD.

Cygwin Unix-like Environment and X-Windows System

Cygwin (/'sig-win/) is a Unix-like environment for Microsoft Windows. Cygwin/X is an X-Windows system that runs on Microsoft Windows. Cygwin was developed by an engineer with Cygnus Solutions. Red Hat purchased Cygnus in 1999. Cygwin/X uses a modified GPL license. For further information, see cygwin.com.

Drools/JBoss Rules Engine

Drools is a popular rules engine that implements the Rete algorithm (and descendents). The current Drools project (version 3) implements a full Rete implementation with optimization and indexing. It also provides field constraints, conditional elements, agenda management, truth maintenance, temporal rules, dynamic run-time rules, functions, global data, and a language-independent engine. It also provides an Eclipse Development Environment plug-in. In October of 2005, members of the Drools project voted to become part of the JBoss Enterprise Middleware System (JEMS). Drools uses the Apache license. For further information, see labs.jboss.com/portal/jbossrules.

Eclipse Development Platform

Eclipse is an extensible development platform. It uses a plug-in architecture that supports any number of programming languages, including Java, C/C++, Fortran, PHP, Perl, Ruby, COBOL, UML2, and Python, among others. The plug-in environment is flexible, allowing diverse development work, such as Wikipedia editing and configuration management. Eclipse was originally developed by IBM and released using the Common Public License. In 2001, Borland, IBM, Merant, QNX, Rational Software, Red Hat, SUSE, TogetherSoft, and Webgain formed the Eclipse Foundation, a not-for-profit corporation to oversee Eclipse development. The Eclipse Foundation presently has over 115 member companies. It is hosting nine major OSS projects and more than fifty subprojects. Eclipse uses the Eclipse Public License. For more information, see www.eclipse.org and www.eclipsemag.net.

EnterpriseDB Database

EnterpriseDB is a relational database system that claims Oracle compatibility, and is based on the OSS database PostgreSQL. It consists of a database server, replication server, migration

[73] See www netbsd.org.
[74] See www freebsd.org.
[75] See www.openbsd.org.
[76] For the BSD family tree, see ftp://ftp.netbsd.org/pub/NetBSD/NetBSD-current/src/share/misc/bsd-family-tree.

toolset, developer studio, debugger, and management server. Sony Online has used EnterpriseDB and converted more than 150 existing Oracle 9i databases used for online gaming [42]. EnterpriseDB was started in 2004 and is based in New Jersey. They have received significant venture capital funding. EnterpriseDB uses a dual-license scheme and the GNU GPL license[77]. For further information, see www.enterprisedb.com.

GCC GNU Compiler Collection

The GNU compiler collection is a set of tools and libraries for developing applications written in a variety of programming languages, including C, C++, Object-C, Fortran, Java, Ada, Pascal, Cobol, Modula-2/3, PL/1, and others. Back-end code generation supports many processors (30+)[78]. The GNU GPL license is used. For additional information, see gcc.gnu.org.

JBoss J2EE Application Server

The JBoss Application Server (/Jay-Boss/) is a Java Platform Enterprise Edition (J2EE) server. It implements the full Java EE set of services, and Sun Microsystems Inc. has certified that it is J2EE 1.4 compliant. It is implemented using Java and is therefore portable to any operating system that supports a Java virtual machine. It has clustering, fail-over, load-balancing, and distributed deployment abilities. It has a large and active user base and is one of the most-used J2EE implementations. Red Hat purchased JBoss Inc. in 2006. JBoss uses the GNU LGPL license. For further information, see www.jboss.org. Note that a number of other OSS J2EE servers are available such as JOnAS[79], and GlassFish[80], and Apache Geronimo[81], among others.

Jena Semantic Modeling Framework

Jena is a development framework for building applications that use semantic modeling to structure and share information. See Appendix E (pg. 93) for a discussion of semantic modeling. Jena is written in Java. It enables Java applications to read, write, and manipulate RDF, RDF-S, and OWL. It also provides in-memory and persistent database storage of classes, properties, and objects. It also provides a SPARQL query engine and a rule-based inference engine. Jena is developed by HP Labs, and uses the Jena license. For further information, see jena.sourceforge.net.

Linux Operating System

Linux is a Unix-like operating system, and has the largest OSS user and developer community. Commercial support is healthy and hundreds of distributions are available[82] from commercial and non-commercial organizations. Linux has been ported to dozens of hardware platforms from watches to real-time application-specific processors to super-computers. A

[77] Appendix D describes dual copyright licensing (pg. 89).
[78] For GCC backend support, see gcc.gnu.org/backends html.
[79] For the JOnAS Java J2EE server, see jonas.objectweb.org.
[80] For the GlassFish Java J2EE server, see glassfish.dev.java net.
[81] For Geronimo, see geronimo.apache.org.
[82] For Linux distributions, see distrowatch.com

relatively large number of applications are available on Linux. Popular distributions include: Ubuntu[83], Mandriva[84], Novell's SUSE[85], Red Hat's Fedora[86], and Debian[87]. Many of the OSS projects listed below are bundled with these Linux distributions. The first version of Linux was placed on the Internet in 1991 by Linus Torvalds, a computer science student in Helsinki, Finland. Torvalds continues to guide development. Linux uses the GPL license. For further information, see en.wikipedia.org/wiki/Linux.

MapServer Internet Map Server

MapServer is a server that renders (on-the-fly) GIS data from a variety of sources into raster and vector images. Supported input data includes SHP, PostgreSQL, Oracle Spatial, ArcSDE, WMS layers, JPG, GIF, and others, along with formats supported by GDAL[88] and OGR[89]. Directly-supported development languages include PHP, Python, Perl, Ruby, Java, and C#, among others. MapServer was developed by the University of Minnesota in cooperation with NASA and the Minnesota Department of Natural Resources. MapServer uses the MapServer license. For further information, see mapserver.gis.umn.edu.

MediaWiki Collaboration Application

MediaWiki is a web collaboration application that enables users to view, edit, and structure content which is available for others on a web server. The most popular example is Wikipedia, a free web encyclopedia (see pg. 3). Wikipedia is an example of the more general wiki (/WICK-ee/) concept. The wiki concept was created by Ward Cunningham in 1994 and was inspired in part by HyperCard style applications. MediaWiki is written in PHP and uses MySQL. Many wiki servers are available. MediaWiki is licensed with the GPL license. For further information, see www.mediawiki.org/wiki/MediaWiki.

Mono .NET Application Framework

Mono is a framework and set of development tools that enable Microsoft .NET applications to be developed and deployed on Linux, Solaris, Mac OS X, Unix, and Windows. It maintains source and binary compatibility across platforms. Mono is based on ECMA and ISO C# and CLI standards. The Mono project was created by Miguel de Icaza in 2001 (de Icaza also created the GNOME desktop for Linux). Version 1.0 was released in 2004. Mono uses the GPL and LGPL licenses. Novell Inc. presently leads the project. For further information, see www.mono-project.com.

[83] See www.ubuntu.com.
[84] See www mandriva.com.
[85] See www novell.com/linux.
[86] See fedora.redhat.com.
[87] See www.debian.org.
[88] For GDAL, the Geospatial Data Abstraction Library, see www remotesensing.org/gdal.
[89] For OGR, see ogr.maptools.org.

Mozilla Firefox Web Brower

Firefox is a web browser available on Windows, Mac OS X, and Linux. It supports more than 40 languages. It provides features similar to Microsoft's Internet Explorer web browser, in addition to a tabbed interface. Firefox is a derivative of the original Netscape Communicator web browser and is the result of Netscape's decision in 1998 to provide an open-source browser (see pg. 9). It is maintained and developed by the not-for-profit Mozilla Foundation. For further information, see www.mozilla.com.

Mozilla Thunderbird Email Application

Thunderbird is an email client available on Windows, Mac OS X, and Linux. It supports more than 30 languages. It supports the IMAP and POP mail protocols, importing from Microsoft Outlook, spam filters, customization, and security features. It is developed and maintained by the Mozilla Foundation. It uses the Mozilla Public License. For further information, see www.mozilla.com.

MySQL Database

MySQL (/My' Ess Queue Ell/) is a very popular open-source database. It supports a large number of operating systems and development languages. MySQL is developed, maintained, and owned by MySQL AB, a Swedish company. The first version was released in 1995. MySQL uses the GPL license. For additional information, see www.mysql.com.

OpenSSH Communications Tools

OpenSSH (Open Secure Shell) is a toolset that provides encrypted secure access between computers. It is an open-source descendent of Secure Shell, which became a proprietary product. It uses the OpenSSH license. For further information, see www.openssh.com.

OpenSSL Secure Sockets Layer Tools

OpenSSL (Open Secure Sockets Layer) is a toolset that provides secure encrypted communication between computers at the socket level. OpenSSL is based on work done by Eric Young and Tim Hudson prior to their being hired by RSA Security Inc. It uses the OpenSSL license. For further information, see www.openssl.org.

Open Office Business Productivity Suite

Open Office is a productivity suite of applications that includes a word processor, spreadsheet, database, drawing program, and presentation package. It is available on Windows, Linux, Mac OS X, FreeBSD, and Solaris. Open Office is compatible with Microsoft Office, but also supports the Open Document Format (ODF) that is being standardized by OASIS[90] as a public document format. Open Office is available in 60+ languages, and is a descendent of Star

[90] For OASIS, see www.oasis-open.org.

Office, which was developed by StarDivision, a German company started in 1986. StarDivision was purchased in 1999 by Sun Microsystems Inc., who initially offered Star Office as a free download for personal use. In 2000, Sun released the Open Office source code under an open-source license and created the Open Office organization to sponsor development and maintenance. Open Office uses the GNU LGPL license. For further information, see www.openoffice.org.

Perl, Python, and PHP Software Development Languages

Perl, Python, and PHP are software development languages that are often lumped together under the general category of scripting languages. They are interpreted procedural object-oriented programming languages used for a diverse variety of tasks, and are sometimes referred to as "duct tape languages" because of their diverse abilities. They are available on almost all operating system platforms. They use the GPL, Python, and PHP licenses respectively. For further information, see www.perl.org, www.python.org, and www.php.net.

PostGIS GIS Database Extension

PostGIS is a GIS spatial extension for the PostgreSQL object-relational database[91]. It adds support for spatial objects (points, lines, polygons, etc.), spatial indexing (r-trees), analytical functions, predicates, operators, coordinate re-projection, and data import and export. It was developed by Refractions Research Inc. The first version was released in 2001. PostGIS uses the GPL license. For additional information, see www.refractions.net.

PostgreSQL Database

PostgreSQL (/post-gress-Q-L/) is a powerful object-relational database. It supports multi-version concurrency control (MVCC), foreign keys, joins, views, triggers, and stored procedures. It supports a large number of operating systems and development languages. PostgreSQL was created at the University of California Berkeley by computer science professor Michael Stonebraker and his students, as a subsequent version of the Ingres database system. The first version was released in 1991. PostgreSQL uses the BSD license. For additional information, see www.postgresql.org.

Protégé Ontology Editor

Protégé is a semantic ontology editor used to create and edit W3C semantic ontology files, such as RDF files. It is used to create properties and classes, and instances of these. It has the ability to generate Java stubs. Protégé is a project at Stanford University and has 52,000 registered users. Protégé uses the Open Content License. For further information, see protege.stanford.edu.

[91] For a summary of OSS GIS, see http://www refractions net/white_papers/oss_briefing/2006-06-OSS-Briefing.pdf

Rdesktop Remote Desktop Client

Rdesktop is a remote desktop client that uses the Remote Desktop Protocol (RDP). It runs on Linux/BSD and is used to connect to machines running an RDP server (e.g. Windows XP Professional, Windows 2000 Server, etc.). It supports disk redirection, sound transfer, and parallel and serial port traffic. Rdesktop was originally written by Matthew Chapman. Rdesktop uses the GPL license. For further information, see www.rdesktop.org.

Samba File and Print Services

Samba provides file and printer sharing services for a number of computing clients, including Microsoft Windows. This enables Linux and BSD servers to transparently act as Windows file servers. Samba uses the GPL License. For further information, see www.samba.org.

VNC Remote Desktop Client and Server

The VNC (Virtual Network Computing) protocol is used by a number of client and server tools to provide remote access to networked machines. It is similar to the RDP approach. VNC was originally developed by AT&T and is licensed with the GPL license. It runs on many operating systems (Linux, Solaris, HP-UX, Windows) and a number of commercial variants provide additional functionality. For additional information, see en.wikipedia.org/wiki/VNC and www.realvnc.com

APPENDIX C:
RELEVANT STANDARDS ORGANIZATIONS AND STANDARDS

The U.S. Department of Transportation has designated the organizations below as responsible for developing ITS standards. More than 100 ITS standards have already been defined. The U.S. DOT ITS Standards web page may be found at http://www.standards.its.dot.gov/

- American Association of State Highway and Transportation Officials (AASHTO): http://www.transportation.org/
- American National Standards Institute (ANSI), Accredited Standards Committee (ASC) X12: http://www.x12.org/
- American Society for Testing & Materials (ASTM): http://www.astm.org
- Institute of Electrical and Electronics Engineers (IEEE): http://www.ieee.org
- Institute of Transportation Engineers (ITE): http://www.ite.org/standards/index.asp
- National Electrical Manufacturers Association (NEMA): http://www.nema.org/
- Society of Automotive Engineers (SAE): http://www.sae.org

The following also specify additional relevant standards:

- Open Geospatial Consortium, an international nonprofit consensus standards organization that developed standards for location and geospatial applications. See http://www.opengeospatial.org/

The standards shown in Table 14 may be of interest for ATMS development. Note that Appendix E discusses semantic modeling standards.

Table 14: Standards of potential use for Caltrans ATMS

Category	Organization	Standards
2D vector data	W3C	SVG
Extensible Markup Language	W3C	XML, XML Namespaces, XInclude, XLink, XPath, XQuery, XSLT
Semantic modeling	W3C	RDF, RDF-S, OWL, SPARQL, Dublin Core
Sensor web enablement	OGC	SensorML, SOS, TransducerML, SPS
Spatial data	Google Inc.	KML
Spatial data and services	OGC	WMS, WFS, GML, OpenGIS Web Services Common Specification
Transducer interface standards	IEEE	IEEE P1451
Content management	IDEAlliance	ICE 2.0 Specification
Web services	W3C	WSDL, XML Schema, SOAP

APPENDIX D:
OPEN-SOURCE LICENSE SUMMARIES

This section summarizes the general open-source license concepts, and provides a brief summary of the more commonly-used open-source licenses. The reader is cautioned to review the specifics of each license directly at the reference provided, in order to understand the implications and obligations imposed by each license. The authors are not attorneys, and make no claims as to the accuracy or completeness of the summaries provided here.

Software code developed by open-source projects is typically protected by a copyright license with special properties such as granting users access to source code. Many open-source licenses stipulate that derivative works must be licensed with the same license as the original. This guarantees availability of derivative source for the community. This type of copyright license is often called *copyleft*.[92] Some of the more popular open-source licenses are discussed below. In general, a copyright is a legal right granted to the author of a creative work to control reproduction and distribution. Copyright law has a long history that originated in England in 1710. The goal of copyright law is to encourage authors to invest in producing creative works, from which society at large benefits. The United States Copyright Act of 1976 grants authors five exclusive rights: reproduction, development of derivatives, distribution, public performance, and public display [69]. In addition, any of the author's rights may be transferred—this is the key to understanding open-source copyright licenses and how they grant users rights. Open-source licenses preserve the ability of users to modify and reproduce software by transferring various rights to users, such as rights to view, modify, and distribute the software. The open-source license is a key innovation that was created by Richard Stallman in 1983 with the GPL (see pg. 10). [93]

A natural question arises—why is a special copyright license necessary? How is using an open-source license like the GPL different from releasing a work into the public domain? All non-copyrighted and expired copyrighted works are considered to be in the public domain. Software written by United States government employees as part of their employment automatically enters the public domain. In general, releasing a work into the public domain means no laws restrict its use, modification, or reproduction. This means that software released into the public domain can be modified *and the modified version can be copyrighted and withdrawn from the public domain*. This is seen as problematic by many in the open-source community and was the initial motivation for Stallman to create the GPL. The GPL requires derivative works to be licensed with the GPL, effectively granting community access to derivative source code. Note that some OSS licenses do not require derived works to be distributed using the original license (e.g. the BSD-style licenses).

Dual-licensing is a key concept in the commercial open-source community. Copyright owners may grant any number of licenses to their work. Commercial open-source corporations sometimes use dual-licensing to simultaneously offer GPL and (for a fee) a commercial license. Examples are EnterpriseDB, MySQL, and Trolltech, among others. For example, Trolltech offers a GPL-licensed version of their software product Qt, which under the terms of the GPL requires

[92] For a detailed explanation of copyleft, see www.gnu.org/copyleft.
[93] For a comprehensive list of frequently asked questions, see www.gnu.org/licenses/gpl-faq html.

a project using Qt to also be licensed using the GPL, opening its source code. Customers may also purchase a Qt commercial license that does not require customers linking code with Qt to divulge their project's source code.[94]

The OSI (Open Source Initiate) defines criteria that software licenses must comply with to be considered "open source." This generally means an open-source license must grant users the right to freely copy, use, and modify the software. It also stipulates that derivate works must be licensed under the same terms as the original.[95] A few of the more popular OSS licenses are briefly discussed below. The OSI maintains a complete list of OSS licenses that meet the OSI's open-source license standards.[96] The OSI web site also contains a listing of each license's text.

Apache License

The Apache license is used by the Apache Software Foundation and their projects, including the Apache Web Server. It is considered a permissive license and does not require publication of source code for derivative works. For further information, see http://www.apache.org/licenses/.

BSD License

The BSD license was originally developed for and used with BSD, a Unix-like operating system released by the University of California, Berkeley. It is the 2nd-most popular type of license. It is considered permissive because it specifies few restrictions. It does not require publication of source code for derivative works. The BSD license therefore is not considered a copyleft license. The non-restrictive nature of the BSD license has enabled a number of commercial products to incorporate BSD software code, including Sun's Solaris, Apple's Mac OS X, and portions of Microsoft Windows. The BSD license is similar to the MIT license. For further information, see http://www.opensource.org/licenses/bsd-license.php.

Creative Commons Licenses

Creative Commons is a non-profit organization started in 2001. It was inspired by the Free Software Foundation and the GNU GPL. However, unlike the GPL, Creative Common licenses are not specific to software, and they offer a number of copyright licenses for sharing images, audio, video, text, software, data, etc. For further information, see creativecommons.org.

GPL License

The GPL (GNU Public License) was developed and released by Richard Stallman and the GNU project in 1989, and is a critically important innovation for the open-source community. The GPL is a copyleft license—derivative works are also covered by the GPL. This quality is often referred to as a viral quality because improvements encourage more improvements, all of which are covered by the GPL. This is also true for code that links with libraries covered by the

[94] For Trolltech licensing, see www.trolltech.com/company/about/businessmodel.
[95] For OSI's Open Source Definition, see www.opensource.org/docs/definition.php.
[96] For OSI approved licenses, see: www.opensource.org/licenses.

GPL. Code that links with GPL-covered code must be licensed with the GPL. For further information, see www.gnu.org/licenses/gpl.txt.

LGPL License

The LGPL (Lesser GNU Public License) is a GNU project license that was originally called the Library GNU Public License. It is similar to the GPL license discussed above, but does not affect the license of code linked with LGPL code. It is therefore more permissive than the GPL license. This enables (and encourages) proprietary applications to link with existing LGPL libraries. Code covered by the LGPL is copyleft protected—derivatives are covered by the LGPL and the associated source code must be released. Examples of LGPL code are the GNU C library and many of JBoss' products. For further information, see www.gnu.org/licenses/lgpl.html.

Mozilla Public License

The Mozilla Public License was created by the Mozilla Foundation to cover foundation software. It is considered a hybrid of the BSD and GPL. For further information, see www.mozilla.org/MPL.

APPENDIX E:
STANDARDIZING DATA FORMATS USING SEMANTIC WEB AND MODELING STANDARDS

This appendix briefly introduces semantic modeling—a new and important technology that is particularly relevant to ITS and ATMS applications. ITS and ATMS applications are fundamentally concerned with sharing information between computer applications that may be located across cities, counties, and districts. Sharing information requires standards. The W3C (World Wide Web Consortium) has developed a group of standards, for defining structured information, that are commonly referred to as the *semantic web* [32,110]. Figure 37 shows how these standards are organized into a so-called *semantic stack*.

The term semantic web can be misleading because the associated standards are fundamentally about syntax, and their use transcends the World Wide Web. The bottom three layers of Figure 37 identify standards for character formatting (Unicode), resource naming and location across a network (URI), and document structure (XML, Namespaces, and XML Schema). For our purposes, the fourth and fifth layers are the most interesting—they define modeling standards used to structure data, and are discussed below. These include RDF (Resource Description Framework), RDF Schema, and OWL (Web Ontology Language), among others. The sixth layer defines a powerful query language known as SPARQL (SPARQL Protocol and RDF Query Language). Higher levels of the semantic stack define functionality that may be useful for ATMS rule-based systems (see the Jena Semantic Web Framework as discussed on pg. 81).

The fourth and fifth layers of the semantic stack define standards for modeling and structuring data. These include RDF, RDF Schema, and OWL, among others. Constructed models are essentially object-oriented and enable the definition of complex classes, properties, and objects. It is important to note that these standards enable much more sophisticated modeling than is possible with traditional object-oriented programming languages (e.g. Java, C++, C#, etc.) or object relational databases. The standards support sophisticated modeling with cardinality constraints, property and class hierarchies, data constraints, and, most importantly, the ability to define and use models distributed over the network. These capabilities are particularly relevant for ITS and ATMS applications, which depend on shared data between agencies and many different types of sensors and hardware.

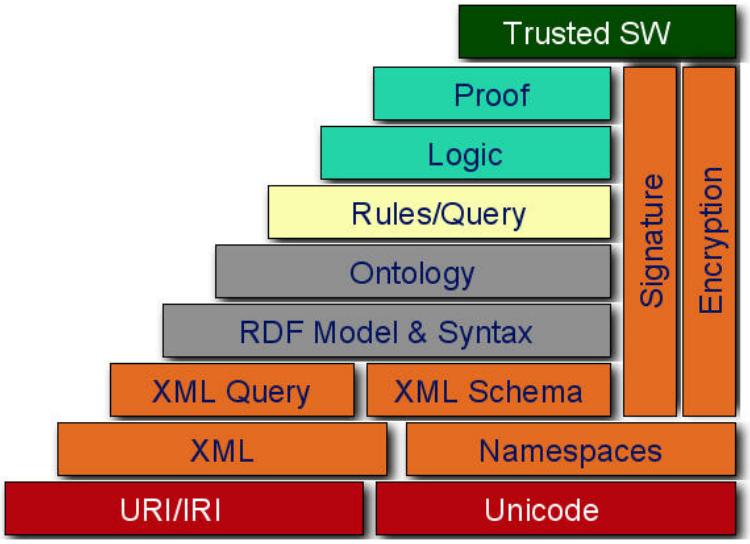

Figure 37: Semantic stack [34]

A practical example of semantic modeling may be helpful. Physical assets such as VDS (vehicle detector station), CMS (changeable message sign), culverts, etc., share many properties such as location, highway number, county, installation date, inspection date, etc. These properties, associated classes, and objects are typically defined using a database or a programming language such as Java, PHP, or C++. A better alternative instead is to define these properties, classes, and objects using a standardized W3C format such as OWL, which is based on XML (eXtensible Markup Language). For example, the code below defines a single property named `countyID` which is a W3C-defined string (via the Uniform Resource Identifier [URI] `http://www.w3.org/2001/XMLSchema#string`), and is used by two classes: `Culvert` and `VDS`. The code below was created using the Protégé open-source ontology editor discussed in Appendix B (pg. 84).

```
<owl:DatatypeProperty rdf:about="#countyID">
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
   <rdfs:domain>
      <owl:Class>
         <owl:unionOf rdf:parseType="Collection">
            <owl:Class rdf:about="#VDS"/>
            <owl:Class rdf:about="#Culvert"/>
         </owl:unionOf>
      </owl:Class>
   </rdfs:domain>
</owl:DatatypeProperty>
```

The `countyID` property defined above is in a nonproprietary format (OWL), is independent of any particular database or language, and would therefore be easy to share with other districts, cities, and states. The `countyID` property defined above can be used by instances of OWL-defined classes. Figure 38 shows a sample UML asset management class hierarchy. The following code defines a portion of this class hierarchy, shown below:

```
<owl:Class rdf:ID="AssetThing"/>
<owl:Class rdf:ID="StationaryThing">
   <rdfs:subClassOf rdf:resource="#AssetThing"/>
</owl:Class>
<owl:Class rdf:ID="VDS">
  <rdfs:subClassOf rdf:resource="# StationaryThing"/>
</owl:Class>
```

The OWL code below (also developed in Protégé) shows an instance of a `VDS` class containing an instance of the `countyID` property:

```
<VDS rdf:ID="vds-313618">
  <countyID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">8</countyID>
  <cal_pm rdf:datatype="http://www.w3.org/2001/XMLSchema#float">80.76</cal_pm>
  <locatedIn rdf:resource="#Caltrans_8"/>
</Culvert>
```

In summary, semantic modeling of a problem domain (e.g. ATMS) uses W3C standards to define classes, properties, and their instances. Classes and properties are typically defined by domain experts and software engineers using ontology editors (e.g. Protégé). These definitions are stored in OWL or RDF Schema files, and define the content and structure of application data. They can be shared with other agencies interested in building interoperable systems. Software developers use the semantic definitions to build applications. Semantic toolkits (e.g. Jena, see pg. 81) can be used to read semantic definitions directly, enable persistent database storage, and integrate with procedural languages like Java. Development of rule-based semantic applications is also supported. One of the key benefits of this approach is that the essence of the system—class and property definitions—is defined in a non-proprietary standardized format that is independent of any particular operating system, database product, or software vendor. This promotes interoperability and reduces lock-in.

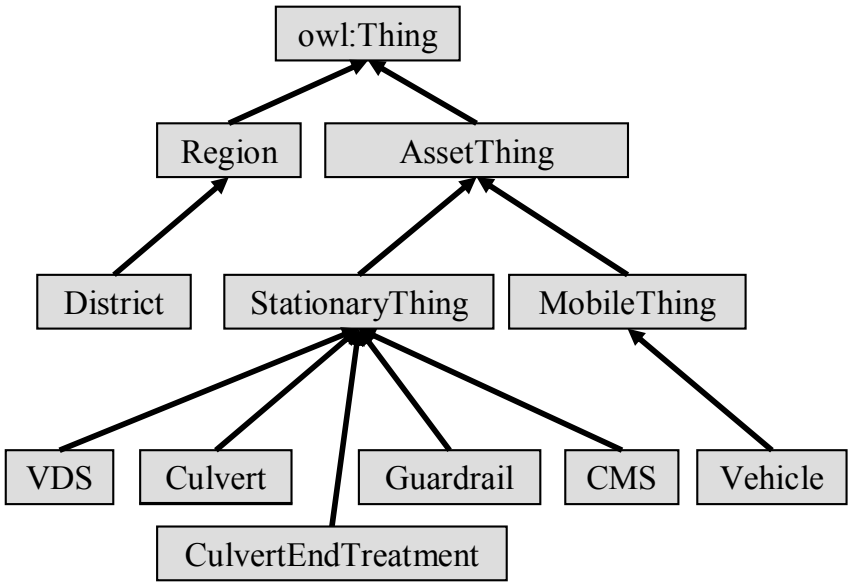

Figure 38: Sample UML class hierarchy