Open Transport Modeling Networking for transportation applications

Marcus R Wigan

Principal. Oxford Systematics GPO Box 126 Heidelberg Victoria Australia 3084, Professor of Transport Systems, Tri, Napier University, 66 Spylaw Rd Edinburgh, Scotland EH10 5BR, UK m.wigan@napier.ac.uk Tel +44 131 455 5140 Fax +44 131 455 5141

Paul Drain

Ridge Technologies, 593 Hull Rd Lilydale Victoria Australia 3140
pdrain@ridge.com.au Tel +61 3 9735 1888 Fax +61 3 9735 3830

Word Count 6745

Abstract.

Transport modeling has developed to a stage where there is a wide range of tools, but a high level of dependence on commercial systems. Recent developments in software and licensing approaches have underpinned the rapid development of robust open source operating systems. In parallel to these developments, formal specifications of data (largely though XML and application-specific XML vocabularies) have begun to emerge for transport-specific applications. At the same time Java has become a widely used development tool. It is argued that these developments allow a new approach to transport modeling and data interchange that can offer higher levels of access and more rapid adaptation to local circumstances and to smaller research groups. The role of global knowledge networking is considered as the underpinning for such an initiative, and the appropriateness of the GPL (General Public License) widely used in open source software is considered.

INTRODUCTION

The first major round of widely used transport models was strongly influenced by the Bureau of Public Roads release of course code in Fortran for a small suite of what were essentially a set of all-or-nothing network modeling and assignment programs. This system provided a starting point for a range of more extensive and powerful commercial and academic modeling systems, examples each being the initial commercial AM Voorhees TRIPS package and the RRLTAP equilibrium modeling system from the UK Transport and Road research laboratory.

This early and influential public initiative was later followed by the more complex family of programs released in ready to run format for the IBM 360 series of computers by the Urban Mass Transit Administration under the title of UTPS (Urban Transport Planning System). This system was less easy to modify and extend as was widely used in its supplied form for a considerable number of years, while commercial systems such as INRO's EMME/2, MVA (UK)'s TRIPS, ITS Leeds' SATURN, and such generally used consulting tools as MINUTP and TranPlan and TMODEL appeared.

More recently different commercial approaches to modeling systems have been delivered, these include,Quadstone (Scotland)'s Paramics highway microsimulation package, Caliper's TransCad system based on a GIS foundation, and the VISUM system from Germany. Several of these newer packages have provisions for product extension by purchases and permit the addition of new algorithms or programs which can utilise the basic data structures and utilities in the full package. Such extensions often gain wider circulation through user groups, but still depend on those wishing to use them securing a copy of the underlying commercial package.

The key common features of the more productive earlier BPR-derived systems are

- Source code availability
- Central source of supply
- Data structure standardization

Data structure standardization was achieved sometimes unwittingly, but was still effective. The BPR a-node, b-node list data structure indexing nodes in a network is still to be found buried deeply within packages still in use today.

There was a long period where innovation in transportation modeling has been largely at a standstill or at least subject only to very slow rates of change. The four step BPR approach is still in use in practice in many places, and the very wide use of a single public sector and widely and cheaply available package (such as UTPS) has faded away to a minority activity, Technically, the use of internal generation of networks and matrices and the forms of models included have altered only slightly over several decades. The key additions have been the construction of matrices from partial data (1980s and 1990s), the construction of networks from GIS sources (1990s) and the inclusion of logit models for choice analysis (mostly in the1990's).

Even so, many of the packages used on the ground still combine a series of external packages such as ALOGIT from Rand Corporation and LIMDEP from Econometric Software to achieve these goals for model estimation and special conflation tools to bridge the gaps between transport network models and spatial information from GIS sources.

The 1990s saw the emergence of microsimulation as a widely used tool for traffic analysis, if not for transport planning. While this has a very long history of traffic applications. For example, Wigan (1) provides a substantial bibliography of a large number of applications even then that included large scale freeway interchanges as well as small traffic intersections).

The emergence of several large scale and far more efficient microsimulation models has now been enabled by the far faster computers now available. Examples in traffic include DYNAMIT from MIT in Cambridge Mass, PARAMICS from Quadstone and SIAS in Edinburgh UK, AIMSUM from Spain, VISUM from Germany and others under active commercial development.

There are also a growing number of other systems with wide public policy impact, such as DYNAMOD from the University of Canberra Australia which is a microsimulation of a full synthetic population of 1% of the Australian population (150,000 records) which is aged over fifty years and targeted at examining the effects of taxation and other public policies on households over time. DYNAMOD simulates fertility, mortality, household formation, income, employment and migration events, but ignores spatial aspects.

INTEGRATION ISSUES

It is increasingly necessary to use a range of models to bring all the different aspects of transport and traffic projects and impacts together. The problems involved fall into several different categories:

- Data formatting and input

- Interchange of data between models
- Interchange between the very different conceptual data structures in different models
- Verification of the internal operation of commercial models when integrated with other types of system (often made even more difficult by a lack of access to the necessary proprietary source code)
- Working with multiple models with very different learning curves and assumptions
- Wide variety of 'add-in' facilities to customize commercial and other closed models for specific purposes

At a practical level families of links between different types of models are emerging. The University of Catalunya group responsible for the AIMSUM microsimulation model has linked network models, Area Traffic Control models and developed links to the widely used EMME/2 assignment system into a package termed GETRAM (3). This is typical of the tactical approach to addressing these structural problems. A similar but more tightly integrated approach is Caliper's development of their microsimulation (TransModeller) as a fully integrated with the GIS and GIS-T data structures and models within Transcad, their GIS-T.

There is also a range of higher-level initiatives designed to alleviate some of this problems. These include moves to formalize the metadata used for transport data inputs. These are not simply a question of adopting a particular Transport Thesaurus, and involves a wide range of international and national conventions for data specifications. Typical examples include the standard codes for commodity specifications (changing these would be a major issue for Bureaus of Census and export categorizations, however awkward they might be for transport applications). Progress in this area is occurring slowly, with library science and transport user and research groups still some way from reaching the necessary common ground.

For data issues Metadata initiatives have begun to be recognized as a necessary component of data modeling in transport and related fields, but this is still far from universally accepted (4). At a different level, the problems of model interworking have been examined by (amongst others) the BRIDGES and SPOTLIGHT (5) projects of the European Union. These projects aimed at defining data structures that would permit data and data structures to be interchanged between different transport models at a higher level than the basis data. This work has shown clearly that the approach has some merit, but cannot easily be made an open set of specifications. XML was used as a framework to ensure a high level of automated transportation and portability, but the resulting GTF (6) vocabulary for model data structures still falls short of the open and transformable specifications that would allow the full interchange processes to be implemented.

The objectives of the GTF initiative were –and are - impeccable, but the context within which it was initially developed was perhaps too limited. We have four domains to consider:

- Data item specifications
- External data structure communication specifications

These two have been addressed in a first iteration by the development of GTF, using the desirable features of XML, where data and structure are intrinsically separable. However the interaction with the transport models needs to be addressed in more detail. Naturally the internal data structures within the models developed and used by different vendors will differ between commercial (or indeed non-commercial) systems, but addressing the external interchange issue does not directly solve the problems of the internal representations and their validation.

To address this aspect would require commercial transport modeling system vendors to supply XML specifications of their operational tools at a level that would, so more than one commercial transport modeling company has advised, give open access to the core intellectual property within their systems. Which indeed it would, but they are all uniformly pleased to enable their systems to read in XML formats and specifications. So we can add two more very desirable objectives:

- XML specifications of internal data structures
- Ability to validate the internal implementations of the models in use.

Only the first is an objective that could be (potentially) addressed by GTF, but the second is closely related to the considerably less well known Dcode (7) initiative also developed within SPOTLIGHT. Dcode addresses the validation and structural information processes that are desirable for transport modeling system users to have in place, and emphasizes the need to have clear links in terms of appropriate application and interpretation. Meeting each of these goals, while highly desirable, would require a degree of understanding of transport models that would be difficult to achieve unless a great deal of diverse expertise could be captured into the proposed information system.

It would be useful to find another route to the same end, where the basis for the models had a more coherent underpinning and common language in terms of structure and data item interpretation. Any such

approach would need to bridge the technical computing infrastructure required to ensure greater transparency and the constructs meaningful to transport model users, implemented on the technical foundation.

This suggests that a profitable route to follow would be to look to a coherent framework for making the best of existing efforts on the underlying technical computing (to build and develop models) and operational experience and expertise in the use of the models thereby produced. This will depend as much on networking between model users as between model builders and designers.

A POSSIBLE STRATEGY

The approaches implied by GTF, SPOTLIGHTs and commercial parties such as TSS are all directed to the external interfacing and the reductions of errors and effort in bringing different types of models together in a coherent and reliable manner. The proposal made in the present paper is to alter the basis for the use, development and validation of transport models to one where economies of scale and more rapid takeup of innovations – and a swifter approach to validation – can be achieved. At this stage such an approach can only be complementary to present commercial initiatives, but there are sound grounds for considering that this could change quite quickly over time.

There are several underpinning technical and computing developments that have made this possible:

- Wide adoption of Object Oriented formulations for models and data systems
- Formal specification languages (specifically UML - Universal Modeling Language)
- Wide adoption of Java in a machine and operating system independent form
- World wide access to common CVS's (Concurrent Versioning Systems)
- The emergence of very high quality and reliable Open Source Unix related operating systems
- Availability of open source data bases and object oriented front ends for existing relational data bases such as Oracle, Informix and IBMs DB2
- Development of commercially viable open source related licensing
- Development of global communities to audit and validate code

Each of these are critical to the proposal of an Open Source Transport Modeling Network. The first six are technical underpinnings but the last are the most critical. The former have a prime role in enabling the networks of human interchange and knowledge base improvement to function in a mixed but global information and transactional economy.

Design and Code issues

Object Oriented formulations for models allow a high level of reuse and this communication of classes of object modeled and types of processes attached to these objects. This is very different to the implementation approaches used in earlier decades, and effectively reintroduce the possibility of common data structure standards, similar to but at a far higher conceptual level than the de facto standards created by the early BPR network data structures.

This capability is not enough on its own, as the formal understanding of each object needs to be very clearly defined for it to be usable by others. While many in the transport field are not yet familiar with UML (8), it is becoming widely used as a formal object description language which can be reasonably easily understood in its diagrammatic form – and which can also be used to generate Java code for the objects directly.

This introduces a high level of underlying rigor into the modeling process. First the objects in the model need to be clearly and formally designed, and then the formal operational description can be generated directly from this. While this does not of itself do more than make formal programmer approaches easier to access, this is a major advance in accessibility of such approaches to program and model design.

Java is designed to be a fully portable language, and draws strongly from the p-code architecture of the UCSD p-system of the 1980s (9). Once again, the speed of modern computers allows the intermediate code approach that underpins Java to be run very swiftly, and a range of efficient compilers to native code exist for various microcomputer architectures, without subverting the portability of the language. Java is naturally object oriented in its design, and furthermore is an ideal match to XML, the data description framework that has been used to implement GTF. Java has been widely adopted as the language of choice for portable systems, and is now commonly used for new transport modeling initiatives.

Another factor that is becoming important is that Java environments are available in Open Source, and run within Open Source operating systems as well as proprietary operating systems. The limited support life of each generation of the dominant Microsoft family of operating systems is atheir firm and published policy, and

the ability to sustain a stable environment for Java code over a longer period may tip the balance towards Open Source operating systems. This does not dictate that development be done under them, simply that a stable and supportable ;environment is and will continue to be available outside.

Concurrent Versioning Systems: local and global

The next step is not purely a technical issue. The growth of global communities supporting and enhancing source code has begun to create very reliable code, and the internet allows common repositories for code development, versioning and access to be maintained in a distributed manner for distributed access. The CVS (Concurrent Versioning System) is a key piece of infrastructure that allows diverse groups and individuals to interact and maintain rapidly developing bodies of computer code – and, just as important, documentation. This is one of the central enabling technologies now used to bring spatially dispersed communities together to address technically demanding tasks.

In CVS, a safe and closed 'sandbox' for work is automatically created when a working copy is checked out for a CVS project using the CVS checkout {project-name} command. The 'sandbox' can be thought of as a controlled area within which CVS can track for changes made to the various source files. The subset of files copied to the developers machine using the 'cvs checkout' command lives on the users machine only and is not touched by the main repository again unless the developer issues a checkin command, where files that have been modified by the user, or that belong to other developers will be automatically updated by CVS to form one basic repository. Thus the developer who lives within the sandbox will stand to gain a lot of benefits of concurrent development. This benefit cannot be achieved for work done outside a sandbox.

Using CVS can also resolve issues with Quality Assurance and other non-code development (the system can also be used with documentation) . In very large projects it makes little sense for the QA team to checkout the entire source into a local sandbox, and then be expected to build near-identical copies of what exists within the repository. Having a large number of developers checking out and rebuilding the same source on similar infrastructure is a waste of resources. .A better way of managing a large number of developers in with a similar build environments in a cross platform environment would be to create a 'branch' CVS on an hourly or daily basis that can be checked out by an automated script that can run several tests (both sanity tests on the code and compile time tests), create ready to use debuggable binaries for the developers and copy these ready made binaries to a centra l location on the network. QA team members can then take the binaries generated by the build team on a regular basis for testing, submitting bug reports or patches back to the developers using the 'comment' option of the CVS 'checkin' facility. This is a formal way in which loosely connected developers can accurately track the progress of outstanding bugs or features within the code under their control without having to constantly request status information from other members of the development community (10).

In very large projects, it does not make sense for the developers to check-out the entire source into the local sandbox. In such cases, they can take the binaries generated by the build team on a regular basis for all those components of the application that is not changed by them and only check-out the parts that are built by the developer.

For example, in a Java modelling project, the build team can keep the results of their last successful build in a standard location in the form of JAR files on the network file servers. Individual developers will use a standard classpath setup that has the network drives mounted on standard paths. Thus, the developers will automatically get the latest version of the files as required by them. To gain the benefits of working within a sandbox as mentioned above, the developer must keep his or her sandbox in sync with the main repository, but does not need to do unless he wishes to.

A group may keep a CVS entirely local to them, there is no necessity to checkin with the other more widely accessible CVS, consequently the commercial and non-commercial work can be managed in a manner allowing effective coordination as desired. A regular CVS update with the appropriate tag or branch name will ensure that the sandboxes are kept up to date.

The tools for the job are not restricted to CVS, although CVS makes distributed and otherwise very loosely coordinated work possible and manageable.

Open source environments

The extensive adoption of the Open Source approach to software development has attracted many parties to contribute databases and other tools, and effective variants on the licensing models have allowed commercial vendors to participate in this process. The niceties of these intellectual property management capabilities are critical for the proposed Open Source Transport Models initiative.

Open Source software is a key component is any future initiative for transport modeling. Issues are beginning to arise that could make working with the dominant Microsoft operating systems and tools increasingly problematical.

- Restrictions to their use in environments excluding Open Source software systems
- Interactions with the GPL
- Withdrawal of Java from current releases of Windows,
- The introduction of non-standard Java 'variants' as part of the .NET strategy.

While all these steps are commercially appropriate for a dominant commercial provider, they introduce a new level of sovereign risk for technical users. In general these risks can be avoided by using Open Source operating systems. As many full Java systems can be run under Windows variants, development can continue using the dominant platform, but delivery may be prove to be best under Open Source operating system.

Open Source software of itself is not a problem for interworking with Windows, it is the terms of the various licences used to manage the intellectual and commercial aspects that cause the complications. The widely used GPL (General Public Licence) presents genuine difficulties when it applies to software than may be merged with software provided under commercial or more restrictive foms of licence.

Open Source software does not of itself introduce a lack of control of a risk to the commercially licensed software on other serves and systems. uncontrolled, or presents a risk to a secured network of commercial systems. The already high and steadily improving standards of reliability and transparency regularly achieved by such widely used and dependable open source systems as the ubiquitous Apache web server and the OpenBSD operating system, which are proven favourites with both private and commercial interests on the internet.

There are two distinct issues here.

- The reliability and transparency of open source software
- The terms of the several different forms of open source Licence formats under which it may be released and used.

Open Source Licenses

There is ample evidence that the reliability and – especially transparency - of open source are a major assets in many projects. The scale of participation in open source activities is remarkable, and the process of mutual testing impressive. A glance at the Sourceforge website (www.sourceforge.net) with its 43,000 projects and just under half a million registered users will show the sale of the activity and the care with which problems and vulnerabilities are addressed, fixed and communicated.

Open source is used far more heavily by technical rather than commercial users . Large communities of interest and support have emerges and matured to deliver networked resources such as Sourceforge. Open Source is now a suitable host for the development, delivery and extension of new generations of transport planning systems. No change in hardware is needed, and multitasking, multiprocessing and multithreading is all available in a mature format if and when it is needed. Two of the largest projects in open source are :

- The Mozilla Web browser: an Open Source version of the Netscape browser, components of which are used in everything from embedded browers for use on GPS and PDA systems .
- The Apple Darwin operating system, which is the core of the new OS X operating system for all Apple computers

Darwin is another large scale and extremely widely used Open Source project. However, Darwin initially had problems with the customized form of license that Apple initiated for it, but this has now largely been addressed and adjusted. The mechanisms that deliver robustness and transparency, and allow standards to be maintained are common to all Open Source software issues under any form of course licencing.

The key problem for all parties is the exact form of the various different licences used to release Open Source software. There are two streams of special interest here: GPL originating from the Free Software Foundation and the BSD/X11 license from Berkeley (11).

The GPL is certainly suitable for university and government use, but can severely constrain commercial parties from close involvement. A BSD/X11 licence does not cause these types of problems, and allows commercial and other parties to comfortably coexist.

This license issue is central to any move to build a network of services and systems for transport planning. There are many well developed and respected systems in commercial use that could be even more effective in such an environment, as long as their intellectual property was secured and revenue streams permitted. One example of a commercial system that can readily coexist with Open Source is the DELTA time

staged land use transport modeling system from the UK. DELTA is currently designed to hand over to any commercial or other assignment package at each time period and then resume. No formal linkage is needed with the assignment package, which could be licensed using GPL, a commercial licence or any other variant without any impact on either – as long as the input and output items and formats were known.

In this case there is no problem with the GPL, as there is no need to access the source code of the assignment package, and only the data formats for interchange are required.… but if an open source assignment package were to be used and integrated tightly into the DELTA package, the form of license used would then have a major impact.

OpenBSD, the secure Berkeley Unix derivative operating system in the Open Source domain, and Mozilla, the Open Source project that is the foundation for current versions of Netscape, have their own formal processes. These comprise two of the most appropriate tried and tested models for Open Source projects.

The choice of form of license is not straightforward, although dual licensing is also possible, and frequently undertaken, A critical issue that has emerged is the need to make any license GPL compatible if at all possible, as this maximizes the likelihood that other developers will support the project. Berkeley have themselves revised the BSD licence to ensure closer compatibility: so too has the Mozilla project.

A variety of factors can influence the decision on how to licence a project . These factors may be purely economic . In such cases the decision if often made under the incorrect assumption that Open Sourced software cannot deliver a fair return on investment for all concerned,. The decision may simply be dictated by organisational policy, or as a question of ownership. This often occurs when a new project is spawned from several other hybrid sources, that may or may not have a 'free' and 'open' licence.

Clearly a lucid guide to Licensing must be a basic element in any open source strategy for transport modeling applications.

When projects reach a stage where code-linking and release to other organizations and to the general public arise, there are two distinct ways to licence the project successfully and avoid complications and violations arising later. Both methods are similar in implementation but have differences depending on

- If developer starts a projec t from scratch. They can choose license you wish use and to maintain and have the people funding the project sign a written agreement confirming this. Or,
- If they based their work - or improved on existing work - on something that exists under a free licence already.

In the later case, If at all possible, it is preferable for them to base their work on an existing program that was released under the GNU GPL. Care should then be taken to inform the project manager, other third-party developers, funding parties and anyone else internally involved with the project that, ``We're not allowed to release the modified version except under the GNU GPL - any other way would be copyright infringement.", this protective clause works because GPL-only code is only usable in GPL projects (12).

If the eventual product (the software) requires both commercial and free distribution (eg. two different subsets of users), the GNU GPL licence can still be used successfully in both cases - as long as the sources remain available to both sets of users - and as a consequence of this availability the modification of these sources is explicitly allowed as well as the redistribution.

One of the more common ways to do this is to have a boxed copy, with manuals, installer and support guide as the commercial version, and a freely downloadable source and binary pair worldwide. One can still sell the boxed copy worldwide and provide sources as a component of the distribution (which in turn, means these sources may be modified and re-distributed as long as their (rather modified) sources include text that describes where the original software was obtained, together with a notice stating that this 'new' distribution was a clearly modified version or subset of the existing one. This approach may appeal to some transport software developers, but in view of the limited market size, other alternatives would probably be preferred.

Software may be developed in conjunction with a third-party who uses software which is not released under a "free" licence (eg. the X11 licence) but is still considered to be free software code – in this case, any derivative works that appear that need to link to the original licence should be 'Dual Licenced', as Dual-license code can be linked with any other free software code (GPL or otherwise).

As one can choose which licence to apply when using the code. If you need to link it with non-GPL code, then it is an option to select the less restrictive BSD license. The code then, is still free to those using it under the GPL licence (eg. those who have downloaded the code from an FTP site, are using it within the parameters of GPL) but can also be linked with non-'free' code in order to make a product that can be redistributed under the alternative licence.

It is important to understand that the 'Copyright' describes the code's ownership. Whereas 'Licensing' is a term of uses for the end user. If there are several licenses, you can choose which one to use. Of course, dual-licensing only works if the copyright owner(s) agree.. It is preferable to get the parties to agree to any licencing structure before commencement of the project).

Auditing processes

One of the major benefits of Open Source development has proved to be the ability to organise auditing and peer quality control on a distributed basis. As much of the new generation of transport tools are being developed at Universities, any approach that allows a large critical mass group of people involved with the overall system will enhance the quality of the outcome of the work, as the internal resources will rarely permit this diversity of expert users and assessors to be any single team.

The operation of the auditing and quality control differs for different versions of the Open Source movement. OpenBSD has a well established auditing process .and only code that has passed this hurdle is integrated into the new releases. OpenBSD relies on a two-tree development model, the '-stable' tree, includes only security and reliability fixes to existing code and the '-current' tree, which are were enhancements, experiments, audit tracking and new features are tested.

Once a '-current' tree has been 'tagged' for release, only patches that are reliability, security or configuration fixes can be added. These patches will then be reviewed by the code maintainer and rejected if they prove to be anything other than a fix (ie. a one-line reliability patch that is coupled with a 100 line code enhancement) or accepted as 'reviewed and cleared' if the code is acceptable.

OpenBSD counters the common "all development is halted in release phase" problem by continuing to allow new code to be added to the "-current" tree at any time - in effect, the "tagged" phase of development is simply taking a snapshot of "-current" and making sure the code within the tree is of release quality.

During the 'tagged' phase, the entire Operating Environment is rebuilt on the various platforms OpenBSD supports - this build phase is entirely automated by scripts, but any piece of code that fails on a given platform generates a fault report which is sent to the code maintainer, the code reviewer and anyone tracking the current build tree (anywhere from 50 to 3,000 people depending on the platform being tested) meaning that a code fault is noticed by many developers, system integrators and users before being subjected to release.

Finally, after a set period of time after the build tests are declared known good on all supported platforms, the 'tagged' release becomes the new -release tree and is released to the community.

This process is already working in a distributed manner for many projects. It would be a small step to have a network of transport model developers operating in a similar manner: some as full participants and contributors, some in a user/review mode, and some to ensure that their commercial tools interface cleanly with what emerges.

The very large Mozilla Open Source prpject has developed practical guidelines (15) on how the auditing process can be done on a distributed basis across a global network of groups and individuals. While the details are not appropriate here, the fact that ones reputation is at stake in ensuring that the process is well and cooperatively undertaken (and often that ones name appears in the source tree) have proved to be powerful motivators.

These considerations are fundamental to any Open Source transport modeling network reaching and maintaining both critical mass and the necessary sustained support on a broad front. The security aspects are not so crucial for applied modeling as for kernel level code, but where cooperative processes or modified kernels are deployed – as could easily be the case in a complex large scale microsimulation model, running on a cluster of machines

Open Source Transport Modeling Network

The subject of this paper is the creation and development and of an Open Source transport modeling process. The key elements have been summarized earlier, and the creation of a collaborative Network of specialists would appear to be a valuable initiative, and one that the present paperspecifically proposes.

Such a process would not start form nothing. There are a number of transport models now being developed in Java, and are currently easily operated under Open Source Operating systems. Two examples are given here, drawn solely on the basis of personal familiarity from the extremely wide range of projects at various stages of initiation, development and maturity across the world.:

- **PedFlow**: An agent based microsimulation model of pedestrian movement and interactions project being developed at Napier University in Scotland (13) in conjunction with collecting fresh pedestrian micromovement data, pedestrian interaction monitoring and measurement. This model also uses XML files instead of a relational or object oriented data bases (14) to hold both data and results, and the match between Java objects and XML data specifications has proved to be very effective.
- **TLUMIP**: A large scale family of models under active development for the State of Oregon. Includes economic activity, household, land use and employment development over time and space, direct micromodelling of 1.8m trips in tours, microsimulation of freight movements. Implemented entirely in

Java, and using a freely available data store (Borland's runtime Jdatastore) to consolidate data and results.

The basic design of these very different models is dependent on object specifications and families of Java classes. At least in the case of TLUMIP, the code is intended by the clients to be made openly available under an open source licence, although exactly which variant is not yet clear.

TLUMIP is just one of the efforts across the globe to improve transport modeling, albeit a large and ambitious one. The next such effort will raise similar questions as to how to make the operational tools widely available, readily built upon and usable with other existing and future data and systems.

It is proposed that an Open Source Transport network be set up, initially at least using the existing Open Source tools and practices, and – again at the first stage at least – based on strict Java and open source licensing. The technical collaboration and remote joint working infrastructures already exist. There are several major studies addressing metadata and new forms of models for freight, passengers, pedestrians and networks are beginning to emerge. The economies of scale and the scope for cumulative gains in transport land use modeling are substantial, and should be appraised as part of any new modeling initiative.

Establishing a basic set of standards and collaboration conventions for auditing and extension would be a small overhead on any current major modeling initiative, and the gains could be perhaps give a payback even within that single first project. There are no spatial boundaries in this new collaborative world, but the mutual trust and personal knowledge of the other participants would be a critical element in establishing the network. Once set up and operating, the learning curve for others to participate would be materially easier and less expertise intensive.

Auditing user code is more difficult than auditing operating system code, but the same pattern of inquiry and testing is appropriate (See Appendix). The procedures and effort levels can be materially reduced from operating system auditing in Java were to be the sole programming language. C and C++ are intrinsically less 'safe', and object oriented design methods provide a further level of simplification for the transport modeling network. Userspace (application) code is more difficult than code in kernel-space to modify and audit for many reasons, but one of the more common traps is to make platform dependant fixes, which may be forgotten by developers later in the development cycle (common amongst 32bit only fixes) which may cause other platforms to break when porting to new architectures like PowerPC, IA64 or portable systems. The Mozilla guidelines (15) offer a practical cooperative framework for this task.

There are major gains to be made in allowing transport modelers to start with a far larger basic framework of tools and starting points than they do now, and this will also allow XML and metadata initiatives to deliver early practical benefits.

This proposal could not have been formulated prior to the emergence of Java, XML, CVS and the Open Source movement. Given these tools are now in place with wide user communities, this paper proposes that the transport modeling community take full advantage of the opportunity for constructive and global collaboration that they can now support.

This proposal does not require any group to adopt more than a cooperative approach to other groups to operate, but the mutual exchanges in the growth and adoption of metadata specifications and collaborative research and development and application work will add up to more than the sum of the parts if some of these principles are espoused.

REFERENCES

1. Wigan, M.R. *Applications of simulation to traffic problems - an annotated bibliography*. Transport and Road Research Laboratory. Library Bibliography LB102/MRW, Crowthorne, UK. 1968.
2. King, A., Baekgaard and Robinson. M. *DYNAMOD-2: An overview* NATSEM Technical Paper 19. University of Canberra (also at http://www.natsem.canberra.edu.au/pubs/tps/tp19/tp19.html)
3. Transport Simulation Systems (TSS). At http://www.tss-bcn.com/products.html
4. Wigan, M.R, Grieco, M., and Hine, J. Enabling and managing greater access to transport data through metadata. *Transportation Research Record* (In Press) 2002.
5. SPOTLIGHTS Thematic network. At http://www.mcrit.com/SPOTLIGHTS/spotlights_folio.htm
6. Neilsen, O., Ruffert, E., and Mandel, B. *Generalised Transportation-data Format (GTF) – data, model, and machine interaction*. Association for European Transport 2001 Annual Conference, Cambridge UK. PTRC, London Proceedings CDrom. 2001
7. Gaudry, M., Neilsen, O. and Tsamboulos, D. *Spotlights TN – WP2 Dcode deonotological code Dcode: a pedigree form requirement (PFR) proposal for European Transformation Information System (ETIS) recognized data and models*. 2001 (at http://www.mcrit.com/SPOTLIGHTS/dcode.htm)
8. Evits, P. *A UML Pattern Language*. Software Engineering Series. Macmillan Technical Publishing. Indianapolis, 2000.
9. Clark, R. and Koehler, S. *The UCSD Pascal handbook: A reference and guidebook for programmers*. Prentice-Hall, Englewood Cliffs. 1982.
10. Slide, D. and Drain, *P. Bridging the gap between developers and users – one step at a time.* ARULUG Annual Conference, Canberra. 2000.
11. Stallman, R.M. *Introduction to the GNU licensing model*. http://www.gnu.org/philosophy/
12. http://www.opensource.org/licenses/bsd-license.html
13. Kerridge, J., Hine, J. and Wigan, M. Agent based modeling of pedestrian movements: the questions that need to be asked and answered. *Environment and Planning B* 3(28), 327-341. 2001.
14. Kukla, R. and Kerridge, J. The use of XML to store tructural data in PEDFLOW. *Software Practice and Experience* (under review)
15. Mozilla Core Review Guidelines. www.mozilla.org/hacking/reviwers.html