TOWARDS AN OPEN ARCHITECTURE FOR REAL-TIME TRAFFIC INFORMATION MANAGEMENT

Mark Dineen, Dr. Vinny Cahill
Department of Computer Science, Trinity College Dublin, Ireland.
Phone: +353 1 608 1795
Fax: +353 1 677 2204
E-mail: mark_dineen@hotmail.com

ABSTRACT

Modern Urban Traffic Control (UTC) and traffic information systems create considerable quantities of real-time data. Unfortunately, this data is then frequently discarded or stored in a proprietary format that makes it difficult to use for purposes such as historical data analysis or other end-user applications. This paper describes an open architecture for the management of real-time traffic information that uses a three-tiered structure in conjunction with the eXtensible Markup Language (XML) in order to allow traffic data to be stored and used by any number of third party applications. An example of how the architecture was used in conjunction with a SCATS UTC system to create Dublin's online urban traffic congestion map is also described in order to illustrate the effectiveness of this approach.

INTRODUCTION

Modern UTC and traffic information systems create considerable quantities of real-time data. Unfortunately, this data is then frequently discarded or stored in a proprietary format that makes it difficult to use for purposes such as historical data analysis or other additional end-user applications. Such data could be put to additional use by extracting it from the proprietary systems and converting it into a universal format that can then be used on as many platforms, and by as many applications, as possible. Creating a standard architecture for such a process also means that end-user applications can be developed that work with a variety of different traffic information sources without having to reinvent the means by which the lower level data is extracted.

This paper describes an open architecture for the management of real-time traffic information that uses a three-tiered structure in combination with the eXtensible Markup Language (XML) in order to allow traffic data from a variety of sources to be stored and used by any number of third party applications. An example of how the architecture was used in conjunction with a SCATS UTC system to create Dublin's urban online traffic congestion map is also described in order to illustrate the effectiveness of this approach.

The paper begins by describing the basic principles behind the open architecture itself. A brief introduction to XML is provided along with the reasons as to why it was used as the method of communication between the three tiers of the architecture. The rationale for using three separate tiers and an explanation of each individual tier's function is discussed. The implementation of Dublin's urban traffic congestion map using this three-tier architecture is then described. The implementation section begins with a brief overview of SCATS and goes on to provide more in-depth details on how each of the tiers and the XML in-between them was implemented. Finally, an evaluation is provided as to how the architecture performed when used to create the congestion map and the future possibilities that exist for extending it.

ARCHITECTURE

The description of the open architecture is divided into two distinct sections. The first section describes the three-tier architecture that contains the primary steps required in transferring the road traffic data from a proprietary system to one that is open and can be used by external applications. This section also explains how the three-tier architecture groups the steps together to create a solution that is scalable. The second section deals with the communication format that was chosen. The eXtensible Markup Language (XML) was selected as it is an open specification that can be used on most platforms and is easily customizable by means of Document Type Definitions (DTD's).

THREE-TIER ARCHITECTURE

A three-tier design was devised in order to separate each of the three distinct processes that need to be undertaken in the open architecture. Figure 1 shows the overall structure of the three-tier architecture. The purpose of the first tier is to extract the relevant data from the system collecting traffic information, e.g. SCATS, and to convert this data into an XML document. The second tier is necessary to allow any intermediate calculations to be performed on the data and, if so required, to store the information in a database. If necessary, the second tier can collate data from a variety of sources as there is no limit to the number of first tier processes that can be running simultaneously. After the second tier is finished processing the data it passes on any data required by the third tier in the form of XML documents. This final tier is where all the third-party applications can reside.

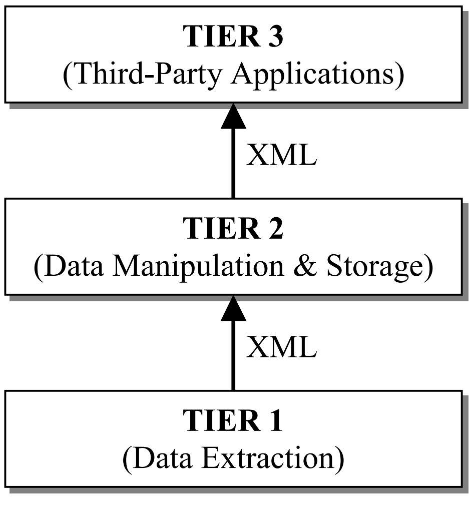


Figure 1. Three-Tier Architecture.

By taking this modular approach to the architectural design several advantages become immediately obvious. First of all, the tiers are location transparent. This means that the physical location of each individual tier is irrelevant as long as the XML files being produced are accessible (e.g. over the Internet). As an example you could have several traffic monitoring systems running Tier 1 applications in different locations around a city. The information from each of these sources could be collated by a central Tier 2 application which creates XML documents that are published on the Internet and are accessible by third party applications from anywhere in the world. The second advantage to this approach is that each tier can be run concurrently so as to minimize the time taken for data to reach the end-user applications. It should be noted however that if the tiers are to be run concurrently then a file locking mechanism for the XML files has to be implemented. Failure to do this can result in "dirty data", i.e. the same XML file is being simultaneously read and overwritten by two different processes. Some possible solutions to this problem are contained in the evaluation section of this paper. Replication transparency is a third advantage. This means that any of the processes can be replicated and run concurrently so as to provide redundancy in the case of failure. Even if one tier fails completely the other tiers will continue to function normally albeit with potentially old data. The architecture also allows for execution transparency. This means that each tier can be run on different hardware and allocated suitable resources according to its requirements. Alternatively, all three tiers can be run on the same platform if so desired. This feature leads to a scaleable architecture as more resources or processes can be added and removed to each tier as the workload changes. Finally, provided that the XML interface between the tiers is not altered, the task of upgrading or changing any of the individual processes is made relatively straightforward due to the modular nature of the architecture.

XML

XML is a specification that was developed by the World-Wide Web Consortium [1] for placing structured data into a text file. By using XML to create a structure for the data, files can be produced that are easily generated and interpreted by different architectural layers. The files are also unambiguous, and avoid problems such as lack of extensibility, lack of support for internationalization, and platform-dependency. In appearance XML looks similar to HTML as both specifications make use of tags (words bracketed by '<' and '>') and attributes. HTML however, specifies what the tags mean (e.g. how the text will appear in a browser) whereas XML uses the tags to delimit items of data and leaves the interpretation of the tags entirely up to the application that reads the file. This allows designers to create their own customized tags thus enabling the definition, transmission, validation and interpretation of data between applications and organizations. A document that conforms to the XML syntax rules (e.g. has no missing tags or attributes) is said to be "well-formed". A document that conforms to the XML syntax rules and follows the guidelines of a Document Type Definition (DTD) that contains definitions of the customized tags is said to be "valid". Ensuring that an XML document is valid inherently carries out basic error checking.

In theory, any type of file protocol (e.g. ANSI text files) could have been used as the method of communication between the architecture layers. There are several reasons however as to why XML was chosen to form the basis of a communications language within the open architecture. Access transparency is achieved as XML files can be retrieved using a unique Internet address in precisely the same manner as HTML files. This property leads to location transparency for the rest of the architecture as the individual tiers can now become location independent. Just like HTML, a single XML file can be read concurrently by multiple clients and as a result, the use of multiple end-user applications becomes possible. Several copies of the same XML file can be produced and stored in different locations thus providing a measure of fault tolerance if required. By using a DTD in conjunction with the XML files it is possible to create a standard that makes the type of data being transmitted easily identifiable by both visual and automated inspection. As XML is a worldwide specification that is supported by most software vendors the individual tiers are not tied to using any vendor-specific software or hardware. Finally, and perhaps most importantly of all, additional tags can be created at any stage in order to cope with new forms of traffic data.

TRAFFIC CONGESTION MAP

In order to gain further insights into the effectiveness of such an open architecture it was decided to develop an end-user interface for a real-life example of a large-scale distributed traffic system that produces high volumes of data. The example chosen was the city of Dublin's UTC system. This system, which is called the Sydney Coordinated Adaptive Traffic System (SCATS), works by measuring real-time traffic flows through several hundred junctions around the city. Using this information SCATS then automatically adjusts the timing of the traffic lights at each intersection on a minute-by-minute basis. A World-Wide Web based interface to the SCATS system was envisaged that could be used to provide the public with real-time information on the current traffic conditions around Dublin city.

It was stipulated that there were three main objectives that any proposed implementation had to be capable of fulfilling. The first of these was that the data from SCATS had to be extracted and converted into a format that could be universally read and understood. The second objective was that an algorithm had to be created and implemented that could take the available data and generate a meaningful measure of road traffic congestion at any given point in time. The third goal was to be able to present this large mass of data to a wide variety of end-users in a manner that would be clear, meaningful and intuitive. As well as these three objectives a number of constraints were placed on any proposed solution to ensure its future viability. These constraints specified that the system had to be scalable, modular, easily configurable, and could not interfere with the running of SCATS. Being scalable meant that no constraints could be placed upon the number of intersections, end-users, or end-user applications that the system could handle. Any proposed solution had to be modular to ensure that the congestion algorithm could be changed at will, that a new version of SCATS could be used without requiring a complete rewrite of all the code, and that extra applications could be added with no difficulty. The solution had to include user-friendly configuration files to allow changes to be made to the intersections and routes that were to be displayed. Finally, for obvious reasons, any proposed solution could in no way interfere with the running of SCATS itself, which meant that only existing interfaces to SCATS could be used. After analysing these objectives and constraints it was decided that the proposed open architecture would, in theory, be capable of meeting all of the above criteria.

SCATS

SCATS was first developed in the early 1970's by the Department of Main Roads, New South Wales, Australia [2] and was first installed in Dublin, Ireland in 1989. SCATS is an adaptive UTC system. This means that it operates in real-time to adjust traffic signal timings throughout the system. SCATS tries to optimize pedestrian, public transport and general traffic flows throughout the network in response to any variations in traffic demand. At the time of writing this paper there were a total of five hundred and seventeen signalized intersections throughout the Dublin area of which more than two hundred and sixty were controlled by SCATS. Before the end of 2001 it was planned that the number of SCATS controlled intersections would have been increased to nearly four hundred.

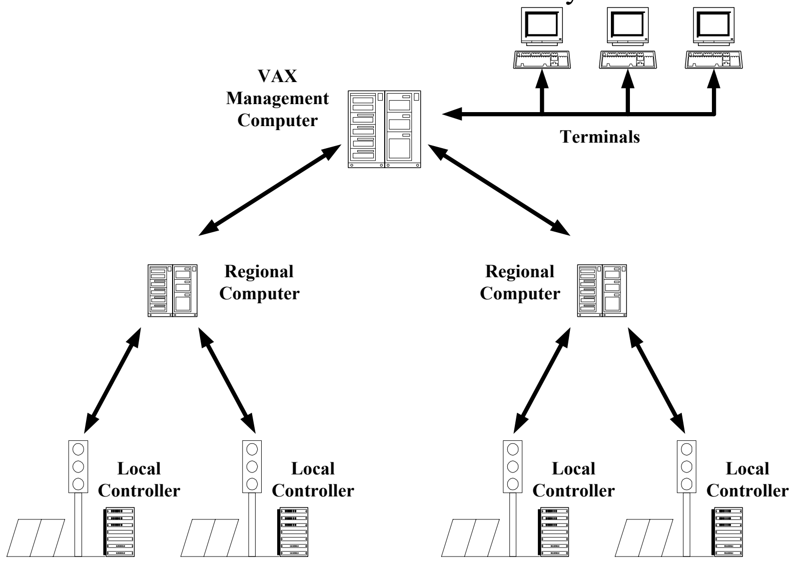


Figure 2. SCATS Physical Architecture.

Figure 2 illustrates the basic hierarchy of the SCATS system [3]. At each intersection there is an inductive loop buried in the ground immediately before the stop-line of each of the main traffic lanes. These inductive loops are connected to the local controller that is situated at the intersection. The local controller is responsible for sending the data collected by the loops to the regional computer, receiving the timing data for the lights back from the regional computer, and implementing the light changes at the intersection.

In Dublin there are five regional computers. The purpose of each regional computer is to analyze the traffic data from up to one hundred and twenty local controllers that are geographically related. Three of the most fundamental values that are determined from the loops in each lane include the original volume (VO), the degree of saturation (DS), and the reconstituted volume (VK). Lowrie [2], details the algorithms used to calculate each of these values. The regional computer assesses the data and calculates the most appropriate cycle lengths, splits and offsets for each of the local controllers that are within its jurisdiction. The results of these calculations are then implemented by each of the local controllers. The purpose of the management computer is to provide centralised monitoring of system performance and equipment status. It also provides for remote access to regional computers by workstation operatives who can view the status of individual intersections through the System Monitor (SM) display and can manually adjust the settings that are sent to the local controllers.

IMPLEMENTATION OF THE DUBLIN TRAFFIC CONGESTION MAP

The open architecture implementation of the Dublin urban traffic congestion map was carried out as shown in Figure 3. The diagram shows how information was collected from two different sources in Tier 1. The reason for this was that the congestion algorithm required data from both the SM display and the SYS.LX (configuration) files of SCATS. These two data sources had to be accessed through two different methods. When this data reached the second tier it was collated and a congestion measure calculated which was then passed onto separate end-user applications in the third tier.

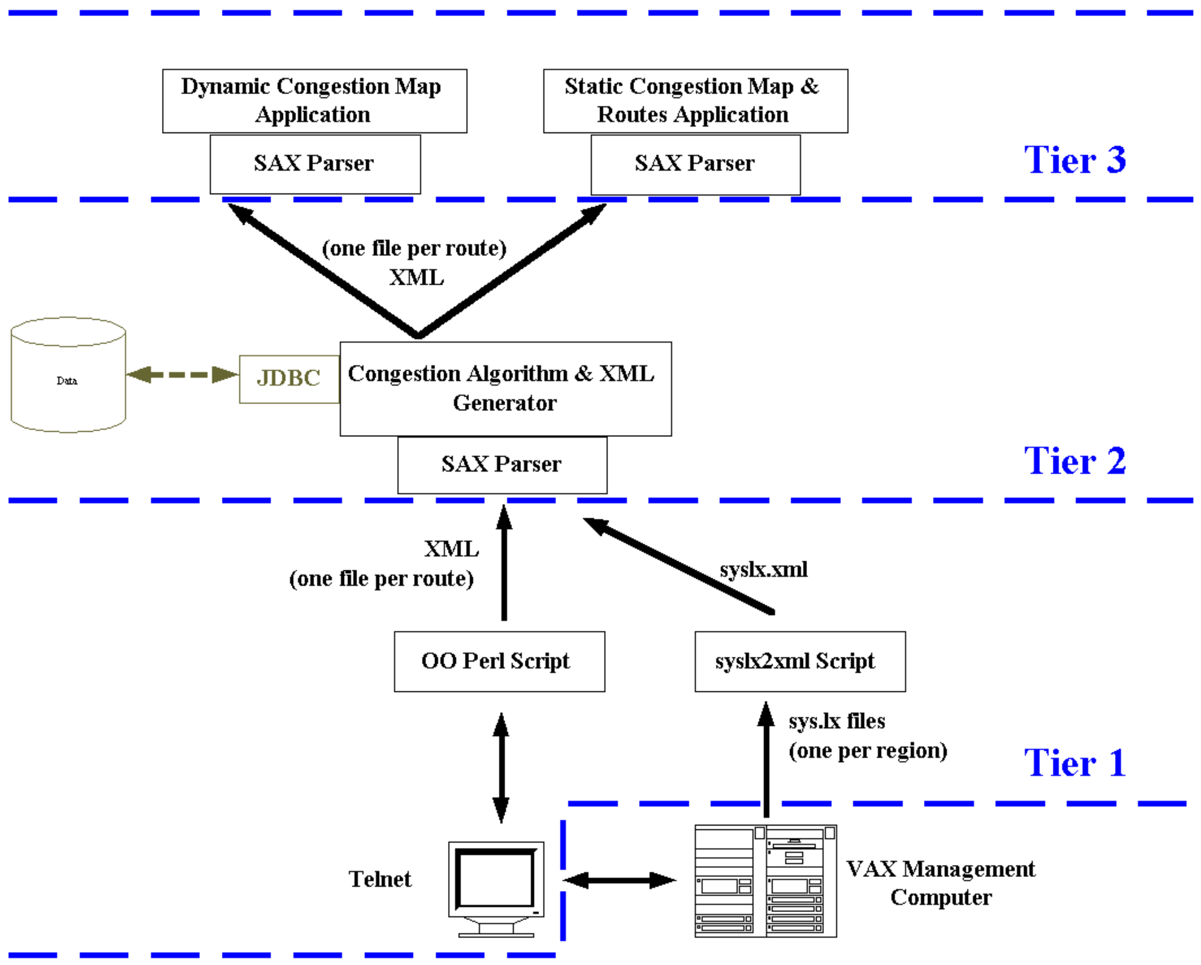


Figure 3. Application of the Open Architecture to the Dublin Traffic Congestion Map.

XML

DTD's had to be designed for every different type of XML file shown in Figure 3. As each XML document is parsed it is checked against its corresponding DTD. If the format of the XML document does not match that of the DTD then an error is flagged, or in the case of the SAX parser that was used in this project, a Java run-time exception is thrown and caught. Figure 4 shows the DTD and a sample extract from the corresponding XML file that was passed between the Object Oriented (OO) Perl Script of Tier 1 and the SAX parser of Tier 2. A separate XML file using the same DTD was created for each route. As can be seen from Figure 4 the contents of the XML file are nearly self-explanatory due to the descriptive tags that surround each piece of data.

```
<!ELEMENT smData (routeNumber*, subsystem*) >
 <!ELEMENT routeNumber (#PCDATA) >
 <!ELEMENT subsystem (intersectionID*, time*,
              cycleLength*,  recommendedCycleLength*,
              strategicApproach*) >
  <!ELEMENT intersectionID (#PCDATA) >
  <!ELEMENT time (#PCDATA) >
  <!ELEMENT cycleLength (#PCDATA) >
  <!ELEMENT recommendedCycleLength (#PCDATA) >
  <!ELEMENT strategicApproach (saNumber*, smColumnData*)>
   <!ELEMENT saNumber (#PCDATA) >
   <!ELEMENT smColumnData (columnNumber, DS, VO, VK) >
     <!ELEMENT columnNumber (#PCDATA) >
     <!ELEMENT DS (#PCDATA) >
     <!ELEMENT VO (#PCDATA) >
     <!ELEMENT VK (#PCDATA) >
```

```
<?xml version="1.0" encoding='utf-8'?>
<!DOCTYPE smData SYSTEM "SMData.dtd">

<smData>
 <routeNumber>1</routeNumber>
 <subsystem>
  <intersectionID>179</intersectionID>
  <time>16:58</time>
  <cycleLength>120</cycleLength>
  <recommendedCycleLength>124
  </recommendedCycleLength>
  <strategicApproach>
   <saNumber>53</saNumber>
   <smColumnData>
    <columnNumber>1</columnNumber>
    <DS>113</DS>
    <VO>19</VO>
    <VK>26</VK>
   </smColumnData>
        . . .
   </smColumnData>
  </strategicApproach>
 </subsystem>
</smData>
```

Figure 4. SmData.DTD (Left) and Extract From Corresponding XML File (Right).

Tier 1

The purpose of the first tier is to extract traffic information from multiple sources and pass the data by means of XML to the second tier where it can be collated. In this specific implementation there were two sources of data, the SM display for the real-time data and the SYS.LX files for the intersection configuration information. It was decided that the best method of interfacing with the SM display would be through the use of an automated telnet session. The advantages of this were threefold:

1. A telnet session would appear to SCATS as just another user so the potential extra load from polling the intersections would be within the SCATS design parameters.
2. Tier 1 would be treated as a standard user on SCATS therefore making it possible to be assigned a very low level of SCATS privileges (i.e. read-only access). This greatly decreases any potential security risks.
3. The data is extracted from SCATS in plain text using the SM display. Receiving the data in plain text greatly simplifies any subsequent data manipulation that has to take place.

It was decided to use an object oriented Perl script in Tier 1 as Perl contains some very powerful string manipulation methods (e.g. regular expressions) that help to simplify the task of removing the relevant data from an SM display. Another advantage of using Perl in Tier 1 is that most of the main methods needed to operate an automated telnet session (e.g. login methods) already exist in an add-on module (telnet.pm) that is freely available on the Internet. The script needed to be object oriented so as to simplify the task of organising such large volumes of data.

A second Perl script (syslx2xml) was created in order to transfer data from the SCATS SYS.LX files to Tier 2 which required this extra information for the congestion algorithm. Each regional computer contains a separate SYS.LX file which contains the configuration details for each intersection within that region. One XML file containing the information from all the regional computers was then created and passed onto Tier 2.

Tier 2

Tier 2 provides a solution for the second major objective (i.e. the implementation of a congestion algorithm). At first glance a simpler and faster running architectural solution might appear to be an amalgamation of Tiers 1 and 2 in the same application. There are three main reasons as to why the middle tier works better when designed on its own as an independent entity:

1. Security – the second tier acts a buffer between the first tier, which has access to SCATS, and the third tier which can be situated on the Internet. From a SCATS security standpoint it is much safer if the third tier cannot directly access the second tier.
2. Modularity – it was a specified constraint that the congestion algorithm and the SCATS data extractor be modular and easily swapped out if required. The existence of a second tier makes this task considerably easier.
3. Scalability – if the load on either the first or the second tier becomes sufficiently large then the two tiers can be placed on separate machines that have greater resources.

This last point becomes even more pertinent as at some future stage the design envisaged the addition of a database (greyed out in Figure 3) that would store and retrieve historical traffic data. To have the data extraction routine, congestion algorithm and database interface all on the same tier would inevitably lead to problems in scalability.
The purpose of the Simple API to XML (SAX) parser is to take the XML from Tier 1 as an input, verify that it is valid, and place the data into an appropriate object model. It would have been feasible to forgo the extra overhead that is involved in using the SAX parser by implementing a customised parser. This was not done due to the excellent error handling methods that already exist in SAX. Once the parser has translated the XML data into the object model of Tier 2, the congestion algorithm produces the congestion measures for the various intersections and this data is then converted into XML files that are passed onto the third and final tier by means of placing them onto a web server. The congestion algorithm produces a rating (in the range 0 – 6) of how congested the approaches to an intersection are. This rating is displayed on the end-user congestion map as lines of varying colours.

Tier 3

The third tier satisfies the last objective, which was to provide a clear and meaningful user interface. There were no major constraints placed upon the design of the third tier as the applications on this level could potentially be run on any computer that has access to the Internet. Figure 3 shows the two third tier applications that were implemented as part of this project.

The first of these was the dynamic congestion map application. This client application, which also functions as a Java applet, was designed to allow a user in any location (where Internet access is available) to view all the extracted SCATS data and congestion measures in a directional dynamic map format.

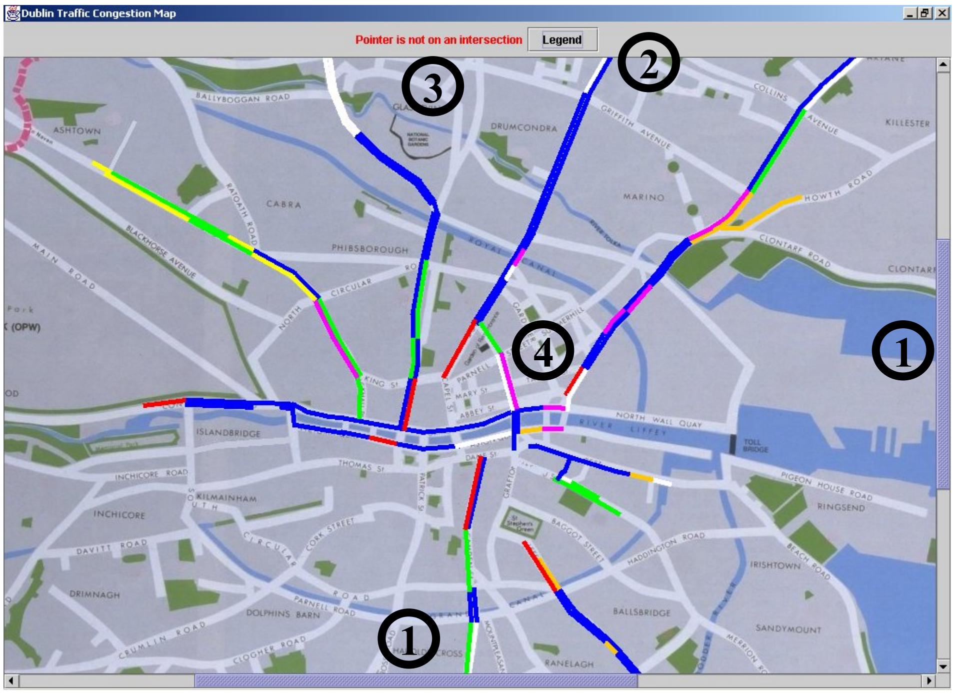
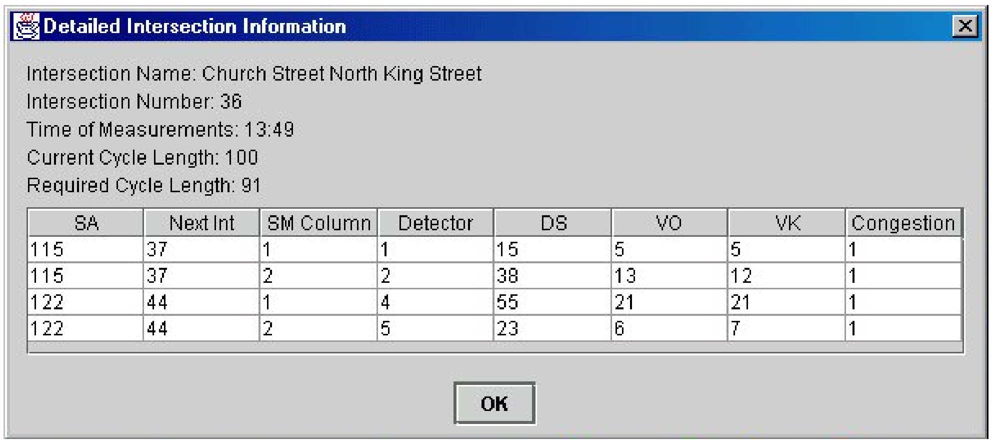


Figure 5. Screenshot of the Dynamic Congestion Map.

Figure 5 shows a screenshot of the dynamic congestion map with the following circled numbers serving to point out some of the features of the map:

1. Scrollbars used to navigate around different areas of the map.
2. Legend button used to display the map legend
3. Intersection name and number that the mouse is currently positioned over.
4. Coloured lines representing differing levels of congestion at intersections along selected routes.

A user can obtain secondary information (e.g. DS, VO, VK, etc) on single lanes by clicking on an intersection. This brings up a dialog box displaying the detailed information as shown in Figure 6.


Detailed Intersection Information

Intersection Name: Church Street North King Street
Intersection Number: 36
Time of Measurements: 13:49
Current Cycle Length: 100
Required Cycle Length: 91

SA	Next Int	SM Column	Detector	DS	VO	VK	Congestion
115	37	1	1	15	5	5	1
115	37	2	2	38	13	12	1
122	44	1	4	55	21	21	1
122	44	2	5	23	6	7	1

Figure 6. Pop-Up Dialog Box Showing Detailed Intersection Information.

The second application shown in Figure 7 is the static congestion map and routes application. The purpose of this is to create static JPEG images of the dynamic map that are regularly refreshed and made available for download on the web server for users that do not wish to download the dynamic application or applet. Static graphics that show the congestion information for each individual route are also provided (see Figure 8). It should be noticed that a time stamp is printed onto all the static images to ensure that the user knows when the data was originally taken from SCATS and whether the images are up-to-date or not.

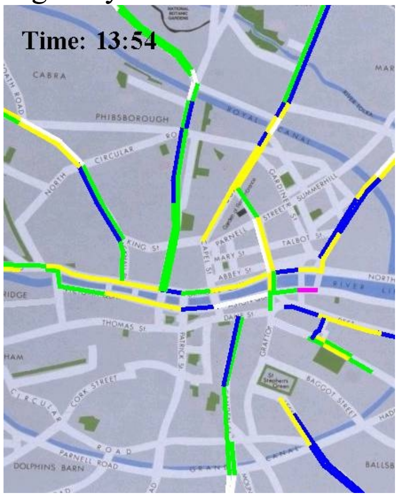


Figure 7. Static Image of Dublin City Centre.

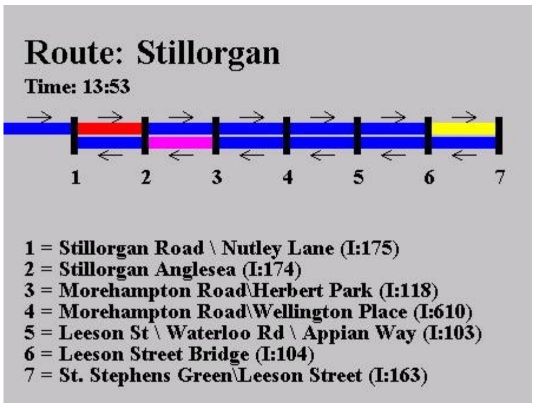

Route: Stillorgan
Time: 13:53
1
2
3
4
5
6
7
1 = Stillorgan Road \ Nutley Lane (I:175)
2 = Stillorgan Anglesea (I:174)
3 = Morehampton Road\Herbert Park (I:118)
4 = Morehampton Road\Wellington Place (I:610)
5 = Leeson St \ Waterloo Rd \ Appian Way (I:103)
6 = Leeson Street Bridge (I:104)
7 = St. Stephens Green\Leeson Street (I:163)

Figure 8. Stillorgan Static Route.

All of these images are then also placed on the web server where they are made available for viewing. The biggest difference between these two applications is that the static congestion map application runs locally within the control of Dublin Corporation whereas the dynamic map can be run on any internal or external machine, provided an Internet connection is available.

RELATED WORK

Before commencing this project a literature review was carried out in order to look at any existing open architectures that were being used in the management of real-time traffic information. During the course of this review very few examples of such architectures were found in the literature. One such open architecture that does exist is the National Transportation Communications for ITS Protocol (NTCIP). The purpose of NTCIP is to provide a *"family of communications standards for transmitting data and messages between microcomputer control devices used in Intelligent Transportation Systems"*[4]. The objectives of this paper are considerably different from those of NTCIP, which aims to provide in-depth details of everything from communication protocols to data object specifications. In contrast, this paper merely seeks to illustrate an architectural approach that can be used in traffic related projects that may have neither the resources, nor the desire, to comply with the detailed standards laid out in NTCIP. In practical terms it was decided that the CORBA-based NTCIP approach would not have been suitable for this type of project due to the following reasons:

1. The steep learning curve involved in using CORBA and complying with the NTCIP standards would have led to significantly slower development progress whereas all of the software tools (Java, XML and Perl) used in this papers approach are already familiar and well understood by many software developers.
2. Integrating the existing SCATS interface with the CORBA-based approach would have involved far more complexity than a simple telnet session.
3. According to the NTCIP Guide [4] a CORBA-based NTCIP approach *"may not be suitable for near real-time applications and loosely coupled systems"*.

An alternative to the NTCIP approach is the Data Exchange in Abstract Syntax Notation (DATEX-ASN) [5] that defines a standard format for traffic messages at a center-to-center level. This standard was not investigated during the course of this project but some interesting further work could involve integrating the DATEX-ASN standard with XML.

EVALUATION

In carrying out the implementation of the open architecture two practical limitations of the design became apparent. Solutions to both of these limitations do exist however and can be used, if necessary, as described in this section. The first limitation was performance, i.e. the speed at which data could be transferred from Tier 1 through to Tier 3. As each Tier operates independently the maximum possible delay is calculated by adding together the time it takes for each loop to refresh its own data. In the Dublin traffic congestion map this amounted to approximately fifty seconds. This delay was of little consequence due to the nature of the third tier applications being used. If however, in a particular implementation, data delay was of crucial importance then it would be possible to create multiple applications within every tier, each of which would handle smaller parcels of work, e.g. a separate process could be spawned to deal with each individual intersection.

The second restraint that should be noted is the issue of concurrency, or more specifically, “dirty data”. Dirty data is what results when corrupted data is read from a file by one process as a result of a second independent process performing a write operation on the same file, at the same time as the first read operation is taking place. Dirty reads can potentially take place at either of the two interfaces between the three tiers. If Tier 1 is writing an XML file while Tier 2 is attempting to read from the same file then it is possible that Tier 2 will receive corrupted data. Tier 3 can potentially receive corrupt data in exactly the same manner if Tier 2 is writing to a file while Tier 3 is reading from it. The three main traditional methods of preventing this from taking place are file locking, shadow paging, and transactional databases. For example, a file locking mechanism between Tiers 1 and 2 would mean that before Tier 1 can begin writing to an XML file it has to first place a lock on it (providing nothing is already accessing the file). This lock prevents Tier 2 from reading the file. Once Tier 1 is finished writing to the file it can release the lock and let Tier 2 access the file once more. The technique of shadow paging is where write operations are not performed on the same file that read operations take place on. The file that the writing takes place on is known as the shadow copy. Once the write operation has been completed on the shadow copy it is renamed as the file that the read operations are to take place on. The final alternative, which only works between the second and third tiers, is where a transactional database in the second tier is used as a buffer. This means that instead of transferring information between the second and third tiers using files, all the information is stored on a database. Any user or third tier application that wishes to access the data would then have to make a request to the web server which would in turn query the database and return the result. The advantage of using this approach is that the transactional database ensures that a file or unit of data that is being written to, cannot be read at the same time. The obvious disadvantage of such a system is the additional overhead and extra level of indirection that is added to the data path. Interestingly, none of these traditional solutions were implemented in the Dublin Traffic Congestion Map. It was found that the parser generated an error if it attempted to read an XML file that was being written to because the incomplete XML file would not match its corresponding DTD. In the case of such an error being thrown and caught the parser would wait until the XML file matched the DTD at which point the file is once again safe to read. This solution, though inelegant, was suitable for the purposes of a proof-of-concept and involved none of the overheads of the other possible solutions.

CONCLUSIONS

This paper described an open architecture for real-time traffic information management that is based upon a three-tiered structure that utilises XML as the communication protocol between the tiers. An example of how such an architecture can be implemented and the benefits it can provide were illustrated through the description of the online Dublin traffic congestion map. The advantages of using a three-tiered architecture are that:

- The system becomes modular with well-defined interfaces so that maintenance and upgrades are made substantially easier.
- Scalability is introduced throughout the system so that extra information sources and end-user applications can be added at will.
- Security can be easily implemented between the tiers.

Using XML as the communications protocol means that valuable data is no longer locked into a proprietary format, any number of third-party applications can be added and the individual tiers gain location transparency. Another significant advantage of using XML is that most modern development environments now provide support for XML as standard.

Some future third-tier applications that are being investigated include integrating a public bus travel-time system with information from SCATS and a Wireless Application Protocol (WAP) interface so that people can receive traffic information on their mobile phones.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the assistance of Dublin Corporation in allowing Trinity College access to SCATS. Brendan O'Brien and Dave Traynor of Dublin Corporation's SCATS Development Unit offered expert knowledge of SCATS while the Director of Traffic and Assistant City Manager, Owen Keegan provided support and funding for the project.

REFERENCES

1 W3C Recommendation: Extensible Markup Language (XML) 1.0 (2nd edition 6 October 2000). Retrieved January 31st, 2001 from the World-Wide Web: http://www.w3.org/XML/
2 Lowrie P.R. (1982) The Sydney Co-ordinated Adaptive Traffic System – principles, methodology, algorithms. *Proc IEEE International Conference on Road Traffic Signalling, London.* pp 67-70.
3 Sims A.G. and Dobinson K.W. (1980) The Sydney Coordinated Adaptive Traffic (SCAT) System Philosophy and Benefits. *IEEE Transactions on Vehicular Technology, Vol. VT-29, No.2.* pp 130-137.
4 The NTCIP Guide. National Transportation Communications for ITS Protocol. (NTCIP 9001, v02.06 December 1999). Retrieved April 30th, 2001 from the World-Wide Web: http://www.ntcip.org/
5 Transport Information and Control Systems – Data Interfaces between centers for transport information and control systems – Part 2: DATEX-ASN (ISO/WD 14827-2)