WORKING PAPER
ITS-WP-98-7

The Use of Object-Oriented Programming Approach in Representing Traffic Noise at the Network Level

by

Tu Ton

March, 1998

ISSN 1440-3501

Established and supported under the Australian Research Council's Key Centre Program.

INSTITUTE OF TRANSPORT STUDIES
The Australian Key Centre
in Transport Management

The University of Sydney
and Monash University

NUMBER: Working Paper ITS-WP-98-7

TITLE: The Use Of Object-Oriented Programming Approach In Representing Traffic Noise At The Network Level

ABSTRACT: Existing road traffic noise models for a single noise receiver are developed with reasonably accurate estimating capability. If these traffic noise models can be incorporated into a network model then the resulting system would be a useful decision support system in transport planning. The key issue in developing a road traffic noise model at the network level is how to structure the basic traffic noise models in such a way that they can be flexibly re-used to construct more complex cases with many noise sources, noise receivers and noise barriers. This paper reports on the use of an object-oriented programming approach to address the identified issue.

AUTHORS: Tu T. Ton

CONTACT: Institute of Transport Studies (Sydney & Monash)
The Australian Key Centre in Transport Management
C37, The University of Sydney NSW 2006
Australia

Telephone: +61 2 9351 0071
Facsimile: +61 2 9351 0088
E-mail: itsinfo@its.usyd.edu.au
Internet: http://www.its.usyd.edu.au

DATE: March, 1998

Introduction

Road traffic noise is an increasingly important environmental impact that has to be managed by road authorities around the world. Significant research efforts have resulted in popular road traffic noise models such as the CORTN model in the UK and the STAMINA model in the USA. However, these road traffic noise models were developed to provide estimate of traffic noise at a point in time (*spot noise level*). If these traffic noise models can be represented in such a way that they can be incorporated into a network planning model, then the resulting system would be a useful decision support system in transport planning in terms of providing the estimates of traffic noise impacts at the network level for any road scheme.

Research in transport engineering has been exploring the use of the object-oriented programming approach (OOA) (eg McGurrin and Wang 1991, Konig and Langbein 1993 and Ton and Black, 1993) as a framework within which many traffic and transport systems and sub-systems can be represented and modelled. Notable applications are in the area of intelligent transport systems (ITS). The use of such tools in representing road traffic noise opens up an opportunity to consider the extent to which there are frameworks which can be used to represent flexibly road traffic noise problems at varying levels of detail.

The paper is structured around five sections. The next section provides an overview of the road traffic noise problem and current issues. Section three contrasts the OOA and conventional representation of road traffic noise. The OOA design framework of road traffic noise is presented in Section four. The evaluation of an object-oriented approach in terms of software quality measures is presented in Section five. The paper concludes with comments on the practicality of OOA in structuring road noise traffic noise from a spot noise level to more complex situation at the road network level.

Overview Of The Road Traffic Noise Problem And Key Issues

Overview of road traffic noise problem

Any road traffic noise model (eg. the CORTN model in the UK and Australia or the STAMINA model in the USA) will require that the road system be divided into segments with each segment treated as a uniform section of the road for which the values of the variables defining road traffic noise are constant throughout. Figure 1 describes the underlying procedure used in CORTN model (UK Department of Transport, 1975 and 1988). Appropriate calculation processing is then applied to every segment in terms of a basic noise level and a number of corrections/adjustments due to traffic characteristics (eg. flow, speed, percentage of heavy vehicle, road gradient and road surface) and the surrounding environment (eg. distance and type of surface between noise source and noise receiver, shielding effect, and reflection factors).

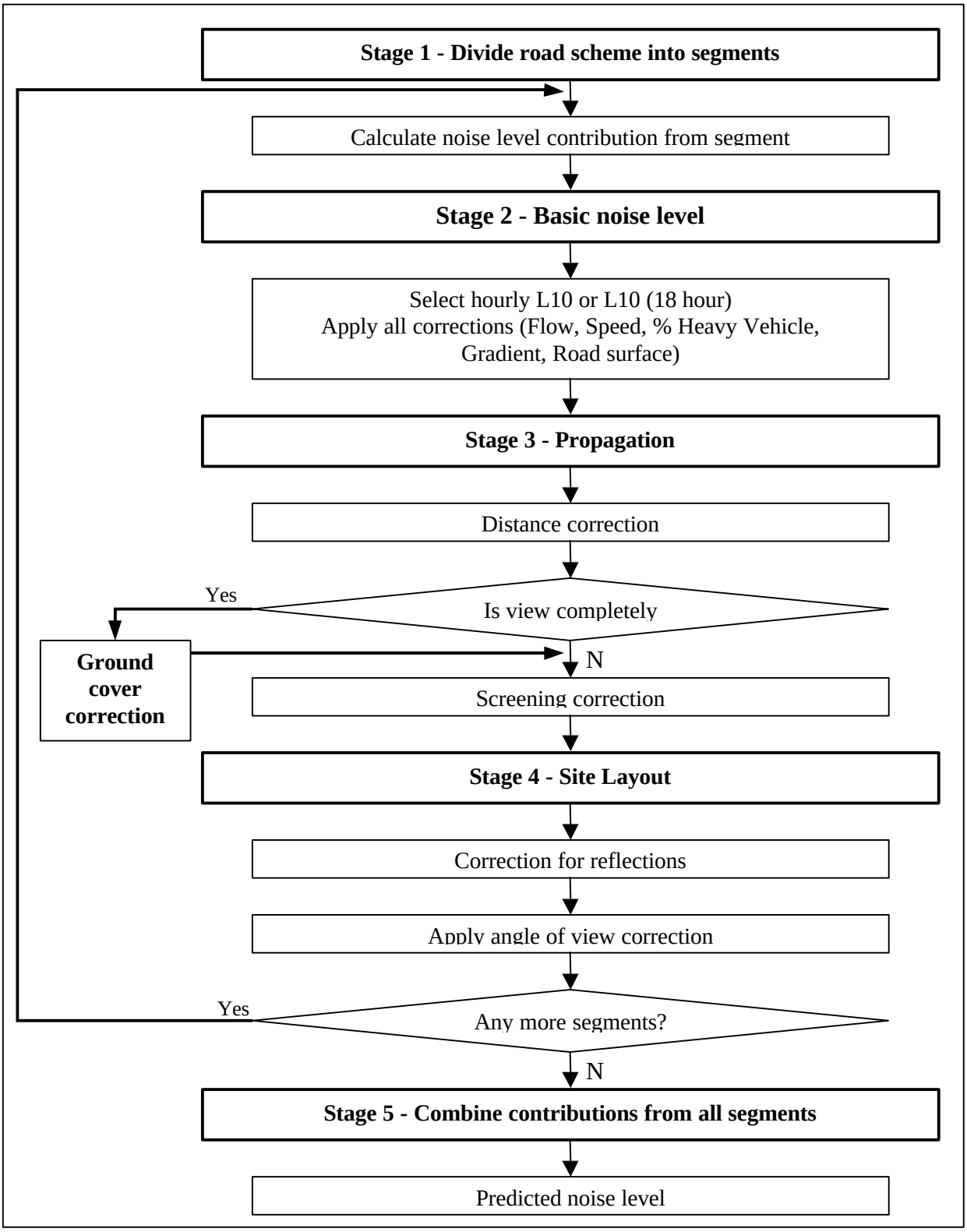

Figure 1: Underlying Procedure of CORTN Road Traffic Noise Model
(Source: UK Department of Transport, 1988, Chart 1, p.38)

The calculation of traffic noise for a single segment represents the basic unit of a traffic noise problem which generally involves the assessment of the interaction (spatial and associated attributes) between three participating objects: *noise source, noise receiver*

and *noise barrier*. In incorporating road traffic noise models into a land-use/transport and environmental model, two specific issues are raised: data integration and model representation. These issues are discussed in the following section.

Key issues of road traffic noise modelling

Data integration and model representation issues of road traffic noise are identified in the context of land-use/transport and environment integrated system. These issues are challenges to road traffic noise modellers:

1. How can data required by road traffic noise models be collected in an automated and cost effective way?
2. How can road traffic noise models be structured and implemented in such a way that they can be used in any configuration of road traffic noise problem at the network level?

The first question about data integration issue can be addressed by using a geographical information system (GIS). With GIS, many different layers of information can be stored, related and used as input/output to and from any integrated models. For example, transport network and zonal layers are used as data for land-use/transport modelling, the output from the network modelling such as estimated traffic flow, speed, etc. can be used as input to road traffic noise model. The spatial modelling capability of GIS can be utilised to store and extract the spatial coordinates (longitude and latitude) for any object from the three objects defining traffic noise - source, barrier and receiver. In addition, the land parcel (cadastral) layer can also be added for more detailed investigation of traffic noise impact to each individual land parcel or household (see Black *et al*, 1996). By using spatial coordinates, GIS is capable of integrating data required for road traffic noise evaluation. However, the lack of efficient support for handling the third component in the spatial coordinates (ie. the elevation), would deter any transport planner in considering to use GIS as a base platform for incorporating traffic noise into a land-use/transport and environment system. Elevation is a crucial geometrical dimension in assessing the spatial relationship between noise sources, noise barriers and noise receivers. This problem is related to the second question which is addressed by this paper.

Different locations (noise receivers) on a network might be subjected to different configurations in their relation to noise sources and noise barriers. The second question demands a modelling approach with flexible representation capability to cope with different cases of road traffic noise. Table 1 lists out 12 possible configurations of traffic noise covering the simplest case of a traffic noise problem which involves a single noise source and a single noise receiver through to a complex case which involves multiple noise sources, noise barriers and noise receivers.

Cases 1 and 2 in Table 1 represent a single segment noise problem involving a noise receiver and at least one noise source and/or noise barrier. The complexity of a noise problem increases with the complex spatial relationship between noise source(s), noise barrier(s) and noise receiver. Cases 4 to 12 in Table 1 represent those situations where

there is more than one noise source or noise barrier or noise receiver. The noise problems for these cases are called multiple segment problems. Each multiple segment problem consists of a number of single segment problems.

Table 1: Major Configurations of Traffic Noise Modelling

Case	Noise source	Noise receiver	Noise barrier	Description
1	S	S	N	Single segment noise problem without noise barrier
2	S	S	S	Single segment noise problem with noise barrier
3	S	S	M	Multiple segment noise problems due to multiple noise barrier
4	M	S	N	Multiple segment noise problems due to multiple noise sources without noise barrier
5	M	S	S	Multiple segment noise problems due to multiple noise sources and single noise barrier
6	M	S	M	Multiple segment noise problems due to multiple noise sources and multiple noise barriers
7	S	M	N	Multiple segment noise problems due to multiple noise receivers without noise barrier
8	S	M	S	Multiple segment noise problem due to multiple noise receivers with single noise barrier
9	S	M	M	Multiple segment noise problems due to multiple noise receiver and multiple noise barriers
10	M	M	N	Multiple segment noise problems due to multiple noise receiver and multiple noise sources without noise barrier
11	M	M	S	Multiple segment noise problems due to multiple noise receivers and multiple noise sources with a single noise barrier
12	M	M	M	Multiple segment noise problems due to multiple noise receivers, multiple noise sources and multiple noise barriers

Note: N - none; S - single; and M- multiple

The key issue in designing a traffic noise model is how to represent the traffic noise problem for single segment in such a way that it can then be re-used to construct a more complex noise problem consisting of many segments. The most complex case is the calculation of traffic noise at the network level where there are a number of multiple segment problems and each segment might have different noise sources, noise receivers and noise barriers. The next section contrasts the two representation schemes provided by two programming approaches: the object-oriented and the conventional.

Representing Road Traffic Noise: Object-Oriented Versus Conventional Approach

Identification of different programming approaches

A programming approach involves a set of mechanisms for performing certain actions (processes, operations, functions, methods, and procedures) on certain data. When laying out an architecture to represent a particular system such as the road traffic noise problem, the modeller is confronted with a fundamental question: what method of system decomposition (partition) should be used? A number of programming approaches are in active use and they provide the modeller with a number of approaches to system decomposition. Although all approaches are represented by some form of language support, not all have advanced enough to have formal support at each stage in the development process. Some of those approaches are: *procedural, logic, access-*

oriented, process-oriented, object-oriented, functional and declarative (McGregor and Sykes, 1992, p.7). Each of these programming paradigms has its supporters and users. Each is particularly suited to a certain type of problem. The procedural programming approach represents the conventional programming approach. The logic programming paradigm is the basis for the so-called *rule-based* systems. The access-oriented paradigm has proved to be a useful technique in structuring user interfaces. Each is a different way of thinking about problems, each uses a different approach to decompose a problem, and each results in different kinds of pieces, procedures, production rules, and so on. The logic paradigm, for example, decomposes knowledge about the problem into a set of discrete rules often represented in a language as "if-then" structures. This paradigm has therefore been used extensively in the development of knowledge-based expert systems.

Due to its popularity in transport computer modelling, the conventional procedural programming paradigm is selected to contrast against the object-oriented programming paradigm. The question of decomposition method is now focused on the fundamental choice: should the structure be based on functions (procedures or actions) or on the data or both? The conventional procedural programming is based on *functions* (actions), whereas the object-oriented approach is based on *objects* (data and the associated functions).

Object-oriented versus conventional programming representation schemes for road traffic noise

To provide a basis for comparison between conventional programming and object-oriented programming, the UK CORTN road traffic noise model is chosen (UK Department of Transport, 1988). Conceptually, at a given reception point, the traffic noise model estimates the noise level generated from one or many adjacent road links, taking into account the shielding effect such as the presence of a noise barrier and other factors such as the gradient of the road link, percentage of heavy vehicles, vehicle speed, surrounding ground condition, retained cut and reflection effects.

A road link is divided into one or more segments which represent homogeneous noise environments. The calculation steps are applied for every segment to calculate the noise level and the associated corrections due to different factors (eg. shielding, traffic composition, reflection, etc.). The final step combines different segments to estimate the combined noise effect.

For each programming paradigm, the discussion will be based on the method of representing the system, the basic architecture and the associated operational characteristics. The following presents the designs of the conventional and object-oriented-oriented approaches.

In representing the traffic noise model, the conventional procedural approach involves the following two steps:

i) Separate data and functions.

ii) Focus on the function abstraction as a unit for decomposing the system. The function abstractions are developed by considering software as a stream of actions. In other words, it is a well defined algorithm that consists of a sequence of steps. Each step is abstracted to a procedure with predefined inputs and specific outputs. The procedures are chained together to produce a reasonably stable flow of control through the program. This results in an architecture that has a very simple but static structure, as typically illustrated in Figure 2.

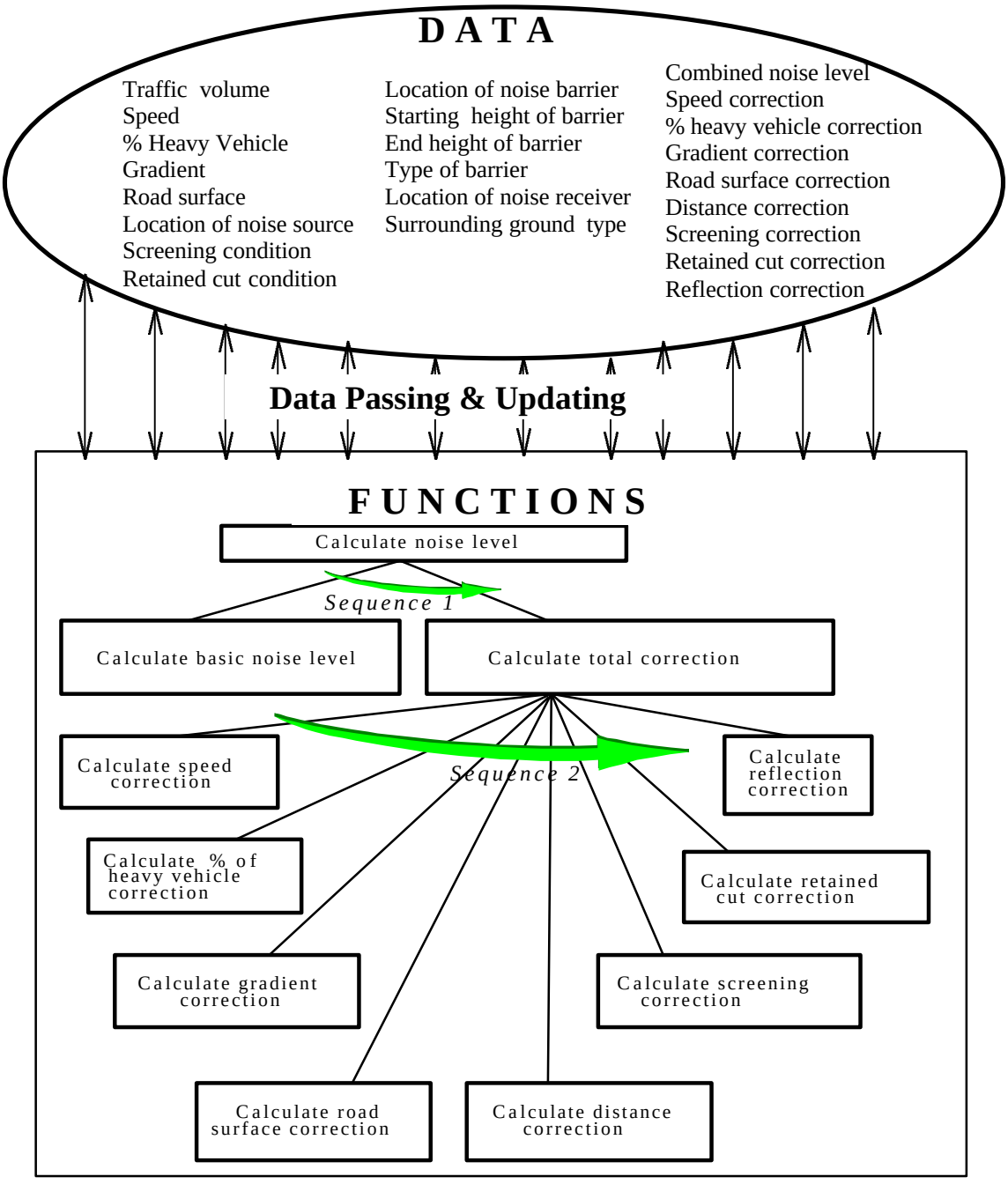

Figure 2: Conventional Programming Design of a Road Traffic Noise Model

In the conventional representation scheme, data and functions are separated. Data are used as function parameters and passed among functions. Functions are the main pieces

of the architecture. In this example, they are placed sequentially although there are possible places for some concurrence. The system is designed by first determining the overall sequencing, followed by defining data to support the basic operations of the procedures.

More specifically, the top level sequence (sequence 1) involves the calculation of a basic noise level and the total correction. In order to calculate the total correction, another sequence (sequence 2) is required to step down to the calculation of every specific correction. In the procedural paradigm the priority is on this stream of actions that constitutes the solution to the target problem. The state of the system is the set of global variables that maintain their value as control moves from one function to another. The resulting application would probably work fine. There has been much support for this approach to system development. However, there have been problems in the development of larger and more complex systems as well as in the construction of user-oriented systems (McGregore and Sykes, 1992, p.10).

In the conventional programming structure of the traffic noise model, the data passing path would become messier ("spaghetti" code) and harder to control as the system becomes more complex. For example, it is not cost effective to modify this design to handle a more general noise calculation program to accommodate multiple noise receivers, multiple noise sources and multiple noise barriers.

The top-down approach employed in conventional programming has two drawbacks. One, it forces the modeller and software developer to impose very strict orders for executing functions. Two, it does not promote software reusability. Although it ensures that the design will meet the initial specifications, it will not promote software reusability. Elements tend to be narrowly adapted to the sub-problem that led to their development; they are not naturally general. Top-down design contradicts software reusability (Meyer, 1988, p.49). Reusable software implies that systems are developed by combining existing components, which is the definition of a bottom-up design.

Techniques to improve the development of large systems have centred around developing data abstractions. The increased attention on the use of an abstract data type has evolved the procedural system development process into a more data-driven approach. Continuing problems with large system development have led to the need to combine the approaches of procedural abstractions and data abstractions. This need has led to the development of the object-oriented paradigm.

The object-oriented paradigm is the result of an evolution in the way problems are decomposed. Where procedural abstraction is the priority in the procedural paradigm, entities - problem domain objects - are the priority in the object-oriented paradigm. In the object-oriented paradigm, the major entities in the problem domain are identified and modelled as the starting point in system development. The objects themselves are the focus instead of the sequence of actions that must be performed on the objects. The objects in an object-oriented system are a unique blend of procedural and data abstractions. The data abstractions representing these entities are a major product of the object-oriented design. The state of the system is maintained through the data stores defined at the core of each data abstraction. These values are maintained for the life of the abstraction. Flow of control is divided into pieces and is included inside each of the operators defined on the data abstractions. Instead of data being passed from one

procedure to another, as in the procedural paradigm, flow of control is passed from one data abstraction to another. The object-oriented design of traffic noise model is shown in Figure 3.

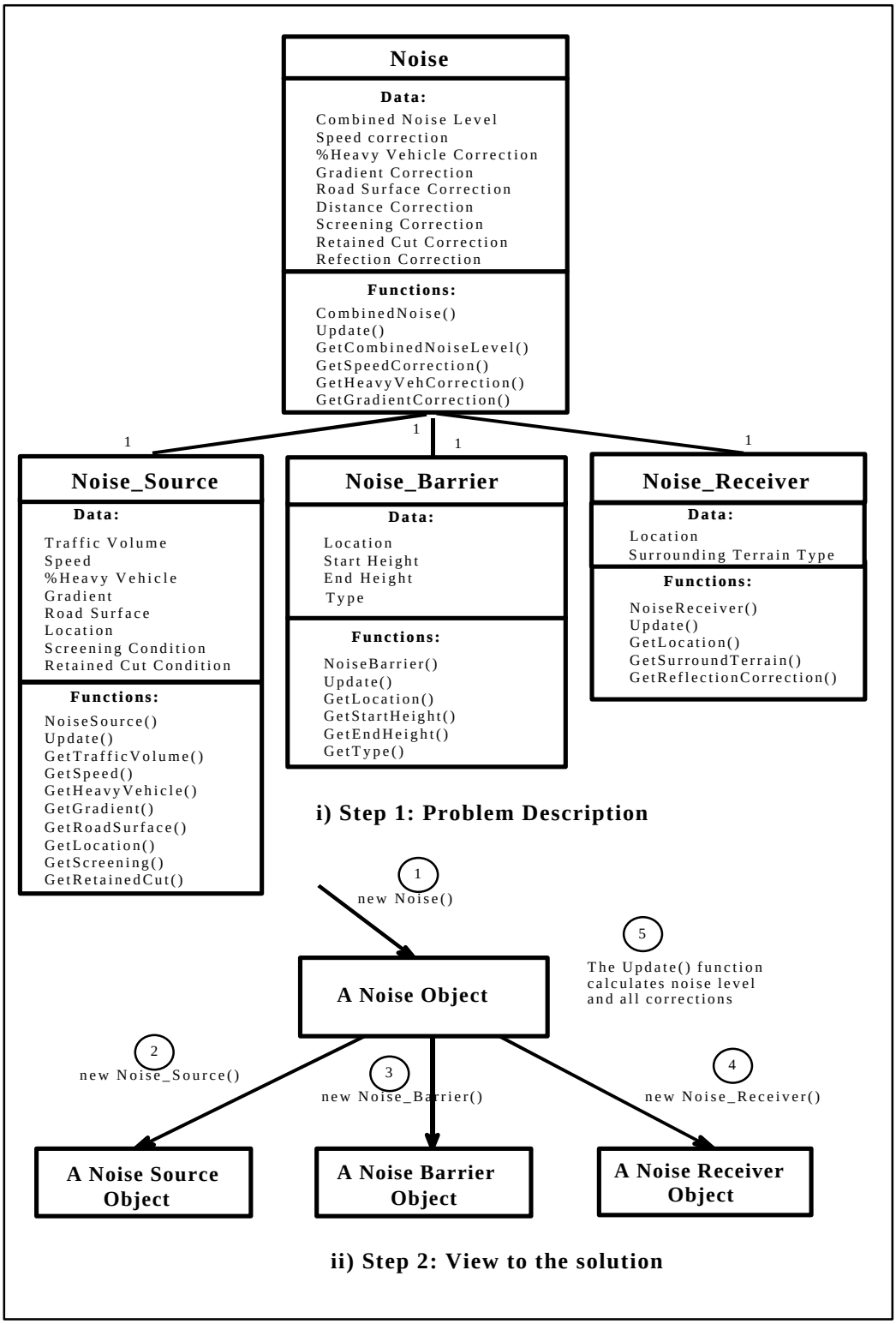

Figure 3: Object-Oriented Design of the Road Traffic Noise Model

As shown on Figure 3, there are two basic steps in the object-oriented design. The first step is to describe the problem (see Figure 3i) and second is to provide the view to the

solution (see Figure 3ii). In Figure 3i, a Noise object has a Noise_Source object, a Noise_Barrier object and a Noise_Receiver object. Each of these objects encapsulate all the basic information (ie. object's data) and the associated functions. A Noise_Source object represents the physical and traffic characteristics of a particular roadway segment adjacent to the noise reception point. A Noise_Barrier object represents a physical noise barrier which is located between the noise source and the noise reception point. A Noise_Receiver object represents a noise reception point (eg. its location and type of surface that surrounds the noise reception point).

A basic difference between the object-oriented and the conventional architecture is the emphasis on "things", physical entities such as the noise source, the noise barrier and the noise receiver. No algorithm is given here because it is really distributed into the various objects. The data passing mechanism used as a means to communicate between functions or procedures in conventional programming is replaced by the message passing mechanism for object communication. As an example, one might want to query about the traffic characteristics of the noise problem; then a relevant message is then sent to the Noise_Source object (eg. NoiseSource1.GetTrafficVolume(), NoiseSource1.GetSpeed(), etc.). From this flexible structure, the next step is to set up a sequence of messages to provide the solution to the problem. The arrows and circled numbers in the diagram indicate messages and message number from one object to another. In order to simplify Figure 3ii, not all of the messages available from each object are shown. The noise problem can be started by creating a noise object (message number 1). In creating a noise object, there is a need to create its components, therefore, three messages (messages 2 to 4) are required to create and obtain information about noise source, noise barrier and noise receiver. The final message is message 5 which is the Update() function which then performs all the calculations for a traffic noise problem.

In summary, in the procedural programming approach, a program comprises a collection of functions that act upon data; this data may be organised into data structures. A problem is "solved" by repeatedly applying these functions. By contrast, the object-oriented approach has a more holistic modelling approach: it models a domain as a collection of computational entities, referred to as *objects*, that stand for the entities that populate the domain. These objects define operations that are invoked as a result of the interaction of these objects with each other. In this approach, a problem is "solved" through the interaction of the objects that make up the problem domain. The object-oriented paradigm gives priority to data abstraction and looks secondarily at procedural abstraction. It is a new technique that has evolved from the increasing interest in abstract data types (ADTs). The entities of the target problem domain are the starting point for system design; implementing them drives the details of system development.

Table 2 summarises the main differences between the two approaches based on their key features. The object-oriented analysis and design appears to be much larger and more complex than the procedural design. This is partly because the object-oriented design provides a view of the solution as well as the problem in one entity- relationship diagram. The result is a description of the problem domain that includes pieces that are much more general and reusable than the pieces that are developed as part of the solution to a specific problem as viewed in the procedural paradigm. While both

techniques result in a workable solution, the object-oriented technique will build for the future enhancement and satisfy the situation at hand.

Table 2: Object-Oriented Versus Conventional Programming Approach

Main Features	Conventional	Object-oriented
Decomposition Method	Function	Object =Data & function
Data Abstraction	No	Yes
Encapsulation of data and functions	No	Yes
Physical Building Block	Sub-program	Object
Physical Structure	Tree of functions	Graph of objects
Existence of Global Data	Yes	Little or No
Logical Building Block	Algorithm	Class Hierarchy
Reusability Support	Limited	Strong

The use of the conventional programming approach in representing road traffic noise models is not flexible enough to cope with frequent changes in system specification and in providing a facility for customising the model to suit specific requirements. Source code reusability represents the most popular form in the conventional software development process. A typical example in transport modelling is the release of the source code for the UTPS package (developed by US Department of Transportation). In UTPS, FORTRAN was used to implement the four-step transport model. The availability of UTPS source code has enabled users (from universities and consultants) to study, imitate and extend the system. However, this type of reusability is not cost effective if one takes a closer look at the nature of repetition in software development. Such an analysis reveals that although programmers do tend to do the same kinds of things time and time again, these are not *exactly* the same things. If they were, the solution would be easy, at least on paper, but in practice so many details may change as to render moot any simple-minded attempt at capturing the commonality. Such is the software engineer's plight: time and time again composing a new variation that elaborates on the same basic themes.

In terms of software quality, the object-oriented decomposition greatly reduces the risk of building complex software systems because they are designed to evolve incrementally from smaller proven systems to which confidence has already been attached. Object-oriented decomposition directly addresses the inherent complexity of software by helping make intelligent decisions regarding the separation of concerns in a large state space. Software quality is improved through the software reuse and software flexibility. In the next section, the object-oriented approach will be used in constructing a framework for road traffic noise modelling.

Object-Oriented Representation of Network Noise Problem

Object-oriented design of road traffic noise

The main feature in the development of an object-oriented framework for road traffic noise is the use of a bottom-up approach. In other words, the framework is built up from a simple case at the spot level (ie. single noise receiver) to complex cases at the network level (ie many noise receivers). A typical spot noise level problem is shown in Figure 4. In this case, the basic requirement is the estimate of noise level at point R (noise receiver) given the contribution of noise impact from the adjacent road (noise source) indicated by source line S_1S_2 and the shielding effect from a noise barrier B_1B_2 (see Figure 4ii for a cross section through S_1R). Inspecting the spatial relationship between noise source, noise barrier and noise receiver reveals that this problem can be structured as a 2-segment problem. One segment is shielded by the noise barrier and the other is not.

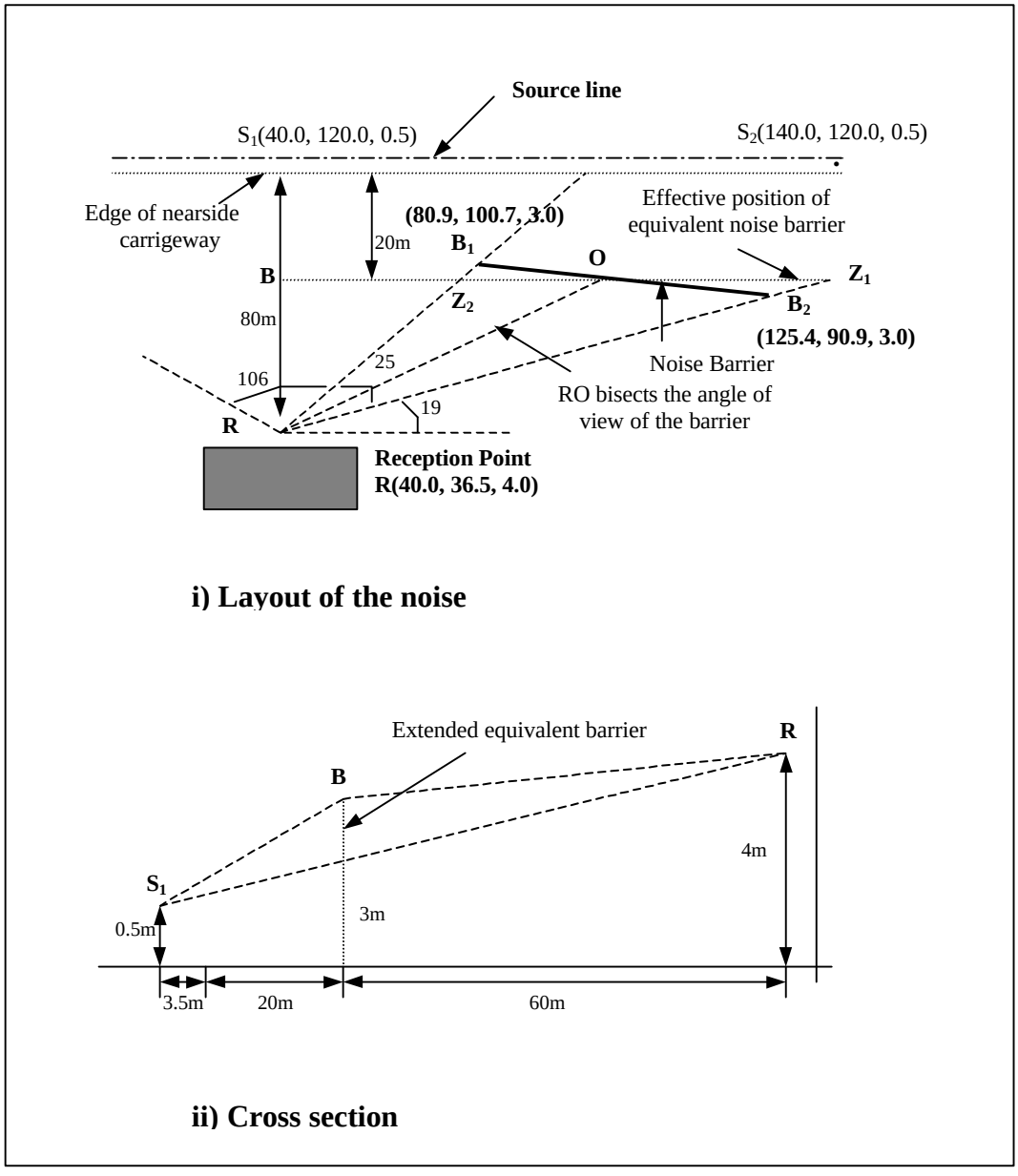

Figure 4: A Typical Road Traffic Noise Problem at the Spot Level
(Source: UK Department of Transport, 1988, Annex 10, p.76)

The three participating objects in this noise problem are represented by three classes (class Noise_Receiver, class Noise_Barrier and class Noise_Source (see Figure 5)).

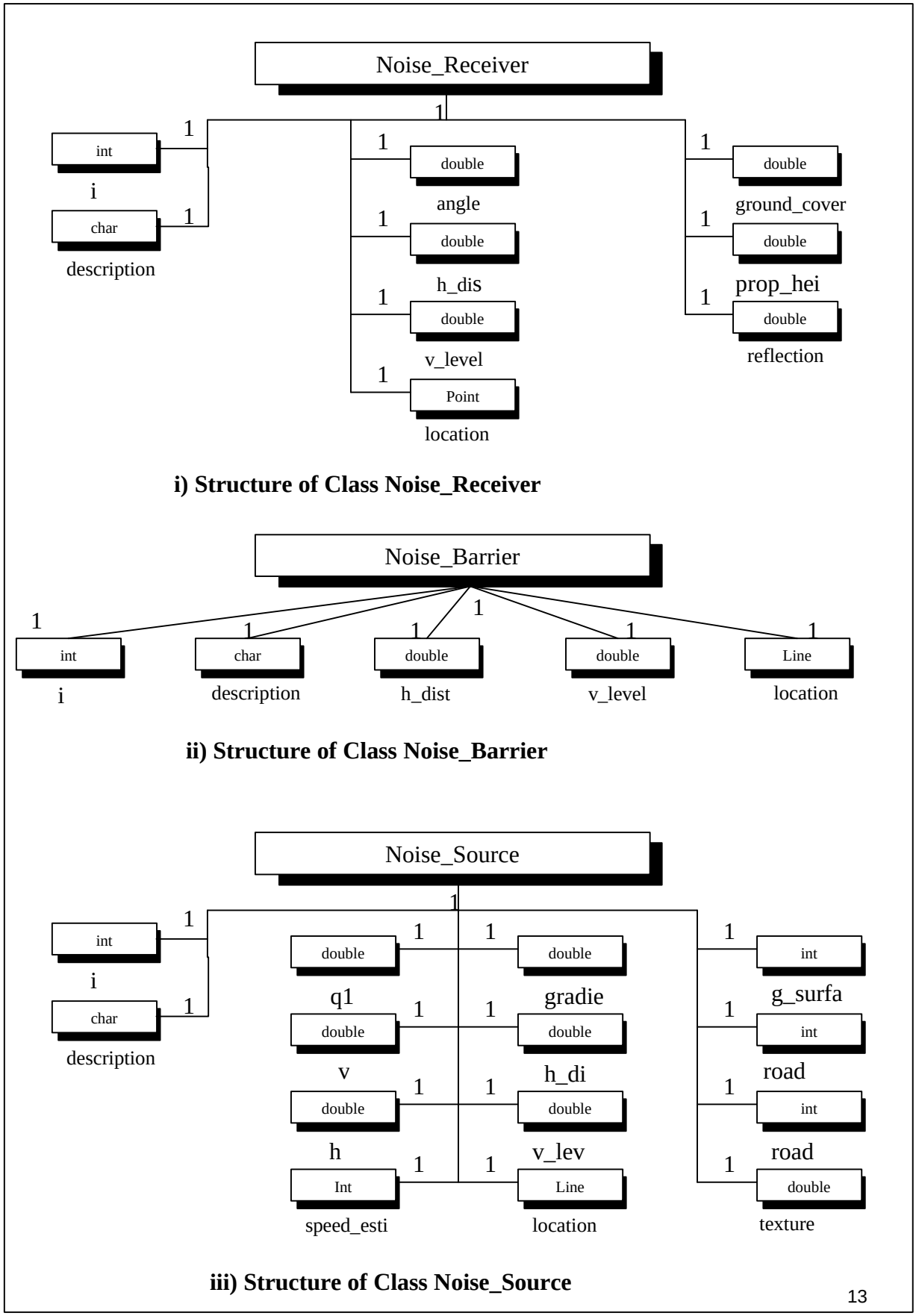

Figure 5: Object-Oriented Design of Three Basic Components of Traffic Noise Model

As shown on Figure 5, these three classes are structured from basic character and numerical types (*double* for double floating point number, *int* for integer, *char* for character string) supported by any commercially available software compiler such as Visual C++. In addition, there are two geometry classes, namely class Point and class Line. These classes are designed as generic classes to provide any spatial modelling support to calculate the distance between noise source and noise receiver or the angle of shielded segment, etc. These geometry classes form part of TRANSOOP which was developed by the author for use as a reusable object-oriented software library in supporting transport modelling. TRANSOOP contains 33,000 lines of C++ programming code implementing 130 basic models identified from the eight supporting domains: land-use, transport, traffic, spatial geometry, environmental impact assessment, mathematics, statistics and computing utility (Ton, 1995).

Once the three key component objects are represented, class Noise_Segment can then be composed to represent a single segment traffic noise problem (see Figure 6). Class Noise_Segment is structured by the three participating objects of classes Noise_Source, Noise_Receiver and Noise_Barrier and all the intermediate input and output data.

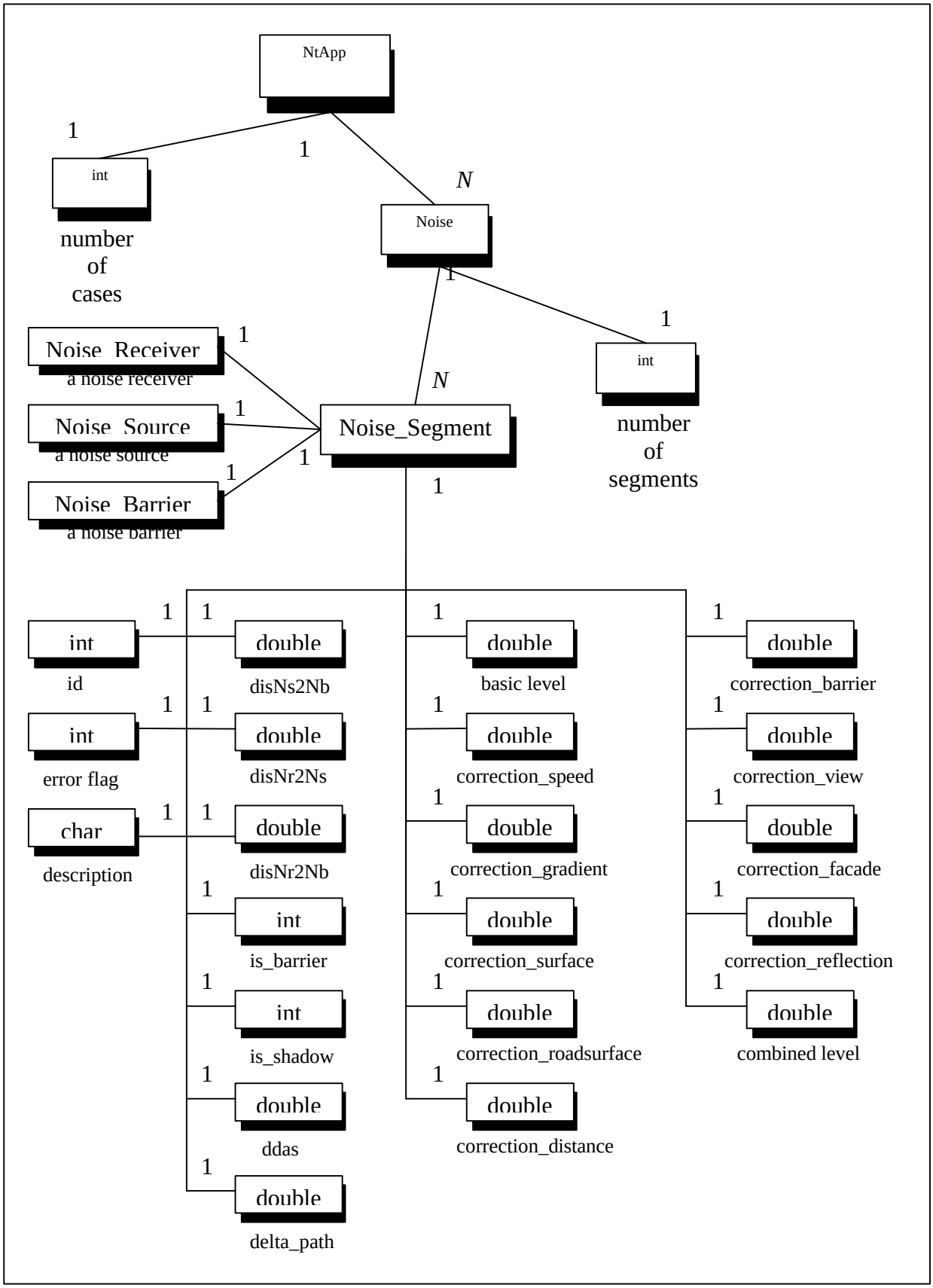

Figure 6: Object-Oriented Design of Traffic Noise Models

The intermediate input data processed by class Noise_Segment describe the spatial relationship between the three object of classes Noise_Source, Noise_Receiver and Noise_Barrier. The output data consists of resulting noise level and the associated noise corrections (adjustment factors). Class Noise_Segment can provide a comprehensive calculation of the basic traffic noise level and the associated traffic noise corrections. Moving further up the hierarchy shown in Figure 6, every case is represented by class Noise. This class is used to represent a complex situation such as multiple segment problems where the noise calculation has to be segmented into various segmented problems which represent the homogeneity of the problem in terms of the relationship between noise source, noise barrier and noise receiver. The top class on the hierarchy shown in Figure 6 is class NtApp. This class represents a general application of any traffic noise calculation. NtApp handles a number of cases which requires the traffic noise impact investigation. This hierarchy can represent all 12 possible cases of traffic noise modelling described above.

Table 3 provides a summary description of this object-oriented representation scheme for road traffic noise problems.

Table 3: List of Object-Oriented Road Traffic Noise Classes

Class Name	Representation	Functional Support
Noise_Source	A physical noise source	Functions for editing and querying noise source data • General information such as id number and description. • Traffic information such as the 18 hour traffic volume q_{18}, the mean speed v, percentage of heavy vehicle hv, the status for estimating the speed, the road surface condition g_surface, road_surface, texture depth. • Spatial information such as location, gradient of the road, vertical and horizontal distances.
Noise_Barrier	A physical noise barrier	Functions for editing and querying noise barrier data • General information such as id number and description. • spatial information such as location of the barrier, vertical and horizontal distances
Noise_Receiver	A physical noise receiver	Functions for editing and querying noise receiver data • General information such as id number and description. • Surrounding terrain information such as the condition of surrounding surface ground_cover, reflection characteristics, the estimation of propagation height prop_height. • Spatial information such as location, vertical and horizontal distances
Noise_Segment	A noise case setup for a single segment calculation (Cases 1 and 2 in Table 1)	Functions for editing and retrieving the three participating objects in a single segmented noise problem (ie. Noise_Source, Noise_Barrier and Noise_Receiver). Functions for calculating basic noise level and the associated corrections (ie. speed, % heavy vehicles, gradient, surrounding terrain, shielding, etc.). Functions for detecting any corrections which contributes maximum or minimum to the noise impact at the segment under investigation Functions for communicating with the three components: noise source, noise receiver and noise barrier
Noise	A case setup for multiple segment calculations (Cases 3,4,5,6,7,8, and 9 in Table 1)	Functions for editing and retrieving Noise_Segment object. Functions for calculating combined noise level for multiple segmented noise problem. Functions for detecting any segment that contributes maximum or minimum to the noise impact.

NtApp	A network noise calculation - a multiple cases setup for multiple segment calculations (Cases 10, 11 and 12 in Table 1)	Functions for editing and retrieving Noise object. Functions for detecting any case which contributes maximum or minimum to the noise impact .

The flexibility of the object-oriented design helps to build a flexible traffic noise model even at the network level. The network noise modelling will be presented in the following section.

Using object-oriented framework to represent the network noise model

The challenge now is how to reuse software developed for the spot level in developing a software system for the network level. As shown on Figure 7, only four additional classes are required to represent the network noise model. They are class NtNoise, class NtSource, class NtBarrier and class NtReceiver. The advantage of the object-oriented approach can be seen at this stage as the software reuse of all other objects of classes. These classes have been constructed and described in previous section.

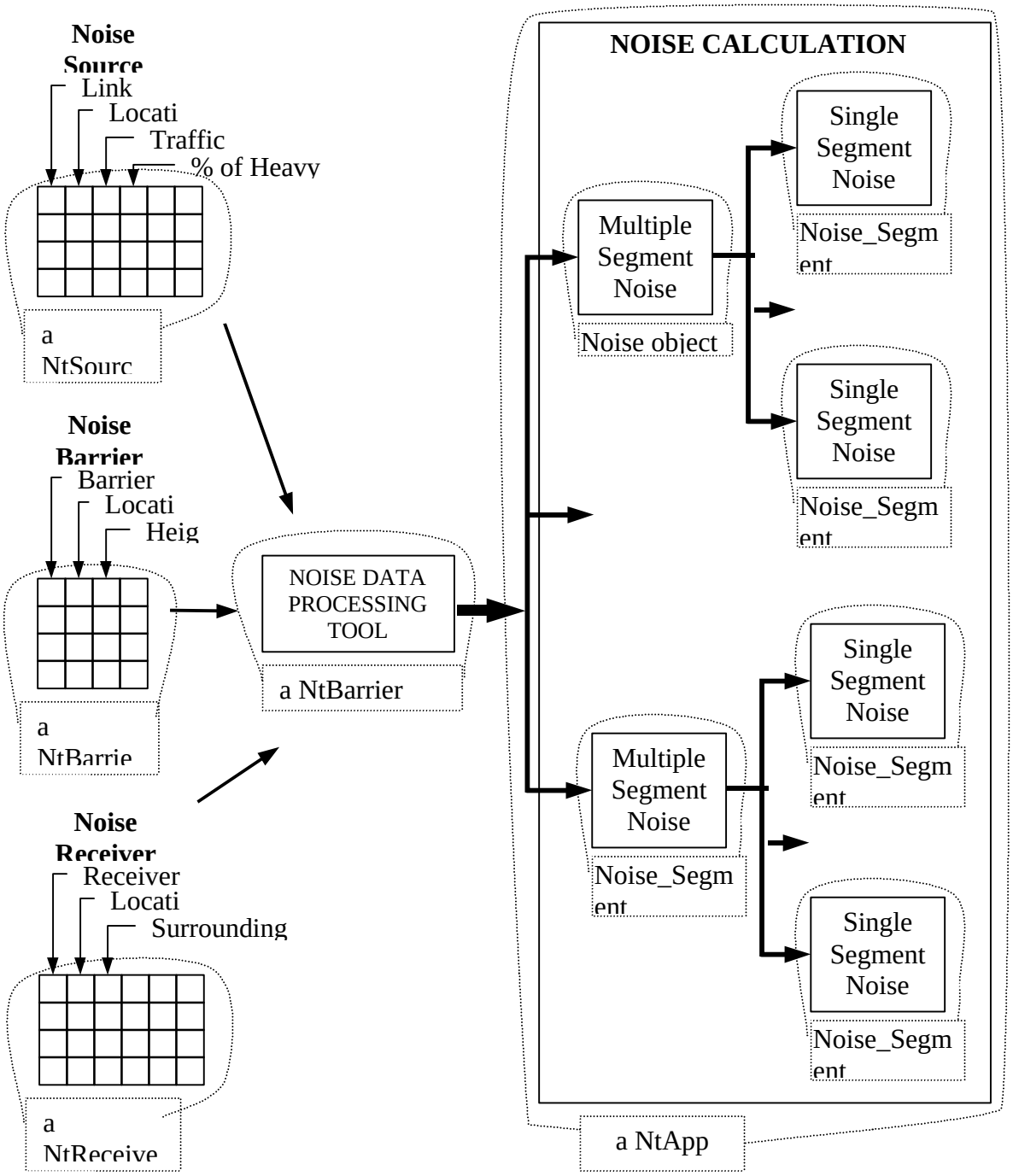

Figure 7: Object-Oriented Design of Network Noise Model

Data processing and calculation are the two major tools required to perform the basic network noise evaluation. The data processing tool is represented by an object of class NtNoise. This object requires the input from three databases: noise source, noise barrier and noise receiver databases. As mentioned earlier, these databases can be available from different layers from GIS. The interfaces to these three databases are represented by the three objects of classes NtSource, NtBarrier and NtReceiver. Not every noise source or noise barrier will contribute to the noise level at a certain noise receiver. Therefore, the main function of the data processing tool is to configurate noise problems and the associated data for the calculation tool. The noise problem configuration

involves a scanning process which looks at each individual noise receiver and checks through the databases of noise sources and noise barriers to select certain noise source(s) and/or noise barrier(s) that might effect the noise level at the noise receiver's location. The selection criterion of noise source(s) and or noise barrier(s) is based on the relative spatial relationship between the three participating objects involved in a noise evaluation problem (ie. noise source, noise barrier and noise receiver).

The calculation tool represented by object of class NtApp consists of a number of multiple segment noise problems as represented by objects of class Noise. Again, each Noise object consists of a number of single segment noise problems as represented by objects of class Noise_Segment. Table 4 summarises the basic representation and associated functional supports of the four object-oriented classes designed for network noise model. These four classes together with the six basic traffic noise classes listed in Table 3 represent a complete framework for handling any configuration of road traffic noise problems.

Table 4: List of Object-Oriented Network Noise Classes

Class Name	Representation	Functional Support
NtSource	A noise source database	Functions for editing and retrieving Noise_Source objects.
NtBarrier	A noise barrier database	Functions for editing and retrieving Noise_Barrier objects.
NtReceiver	A noise receiver database	Functions for editing and retrieving Noise_Receiver objects.
NtNoise	A network noise data processor	Storing and retrieving the 3 databases of noise source, noise barrier and noise receiver. Configure specific noise problems and the associated data by checking the spatial relationship between every noise receiver, noise source and noise barrier on the network.

The application of the object-oriented design of network noise model is demonstrated in a worked example (see Figure 8). This is a hypothetical residential block surrounded by four road links with different traffic characteristics and a noise barrier. The spatial data is in the format of x, y and z representing the 3-dimensional noise problem. The four road links are defined by the four points (S_1, S_2, S_3 and S_4). The noise barrier is defined by two points (B_1 and B_2). The task is to estimate the noise level for a grid of four points (points R_1, R_2, R_3 and R_4) with and without the existence of a noise barrier.

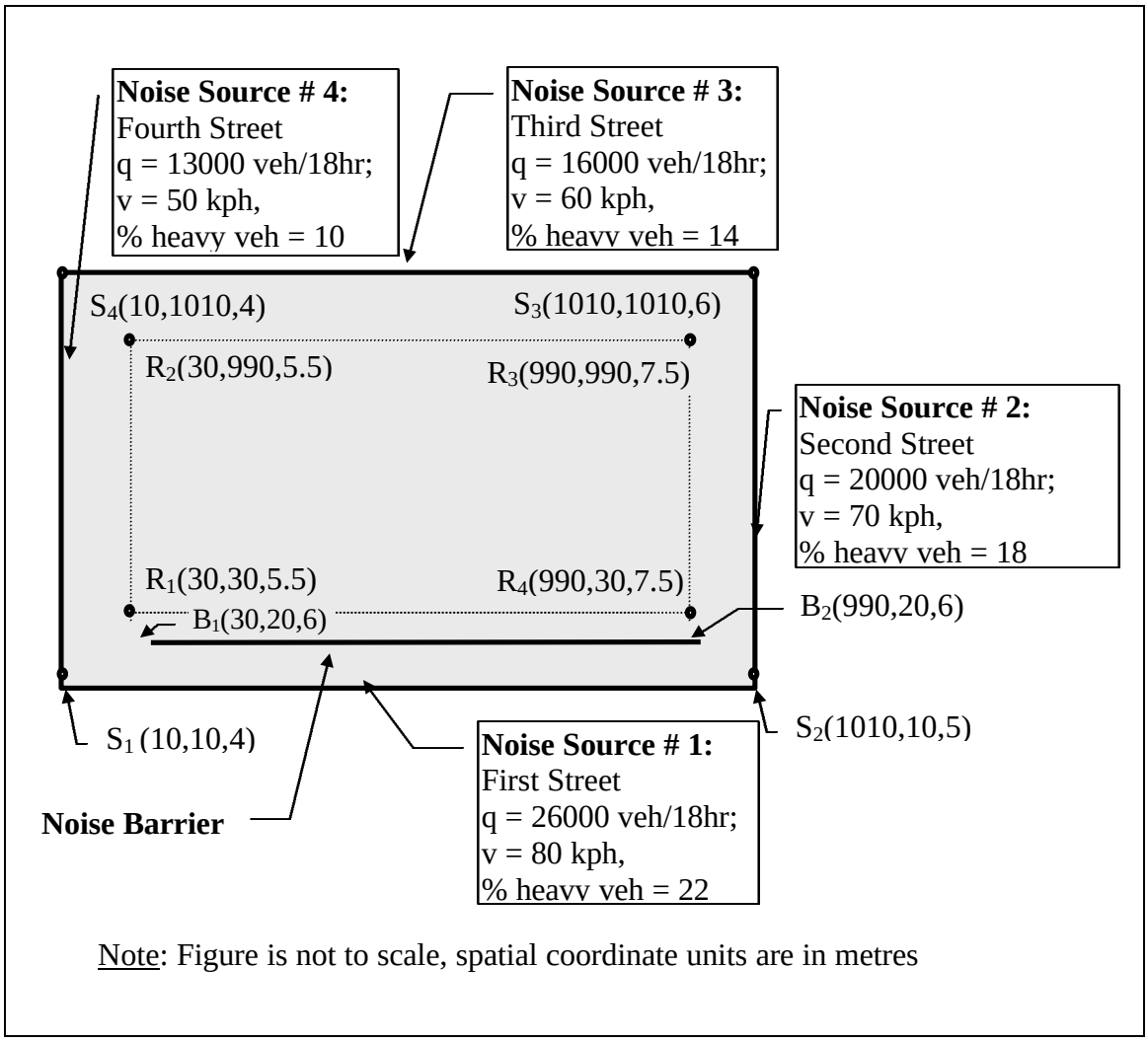

Figure 8: Sketch of a Hypothetical Residential Block for Network Noise Evaluation

Table 5 shows the output in terms of L_{10}(18hour) noise level estimated for the four reception points on a hypothetical block surrounded by four road links with and without the existence of a noise barrier. A manual noise configuration and calculation for the four points confirm the 100 per cent accuracy of the network noise model (implemented in C++ programming language) for both scenarios (with and without a noise barrier). Inspection of the output data for both scenarios found that with the installation of a 2 metre high noise barrier, the noise levels for points R_1 and R_4 have been cut by 4 dBA and 1.2 dBA, respectively.

Table 5: A Summary of Network Noise Evaluation for Worked Example

Scenarios	**Estimated Noise Level** L_{10}(18hour), dBA			
	at Point R_1	at Point R_2	at Point R_3	At Point R_4
Without noise barrier	74.91	74.60	76.95	77.11
With noise barrier	70.97	74.60	76.95	75.94

The integration of this network noise model with GIS and other land-use/transport system is also possible. The functional arrangement of the integration between network noise (TNETNOISE) and GIS is shown in Figure 9.

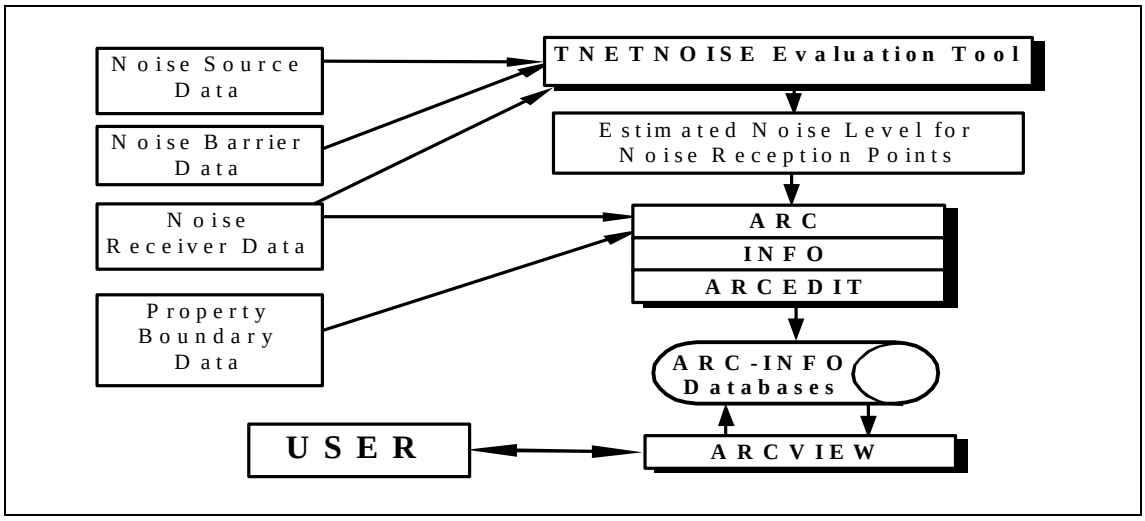

Figure 9: Integrating A Network Noise Model with GIS

Four sets of data provide the input to the application program: noise source data; noise barrier data; noise receiver data; and property boundary data. The noise source data contains the link-based information in terms of location and traffic characteristics, such as traffic volume, and the percentage of heavy vehicles. The noise barrier data contains the information of noise barriers in terms of location and physical dimension, such as the barrier's height. The noise receiver data contains the information of noise reception points in terms of location and any surrounding terrain conditions such as paved or unpaved surfaces. These three noise-related data are the main input to the TNETNOISE.

The role of TNETNOISE is to provide data processing and noise calculation. The output from TNETNOISE is the estimated noise level for each noise reception point. This output is fed into the GIS ARC-INFO (developed by ESRI, 1992) together with the associated noise receiver data and the property boundary data to provide a basic graphical database system for network noise evaluation. Five major tools of ARC-INFO are employed: ARC, INFO, ARCEDIT, ARCVIEW and ARCTOOL. The ARC tool handles the spatial modelling task. The main task in this case study is the generation of noise contours given the spatial data from noise receiver data and the noise level from the TNETNOISE evaluation tool. As a database management system, INFO tool complements the ARC tool in handling aspatial data attributes of the network noise problem such as the aspatial data of noise source, noise barrier and noise receiver. ARCEDIT tools help to edit the property boundary information. ARCVIEW is the front-end of the TNETNOISE system where it helps the user query the network noise database as well as viewing noise contours over the study area.

Detail of this integration applied in a real world case study is given in Black *et al*. (1996). The development of the network noise model demonstrates the feasibility and flexibilty of using an object-oriented approach in developing analytical tools for

environmental impact assessment such as the traffic noise as well as the integration with GIS technology. Formal software quality evaluation of object-oriented designs for road traffic noise models are presented in the next section.

Evaluation Of Object-Oriented Representation For Road Traffic Noise

The evaluation of the object-oriented representation of road traffic noise is based on currently recommended quality measures of general software systems (McGregor and Sykes 1992, Booch 1991, and Meyer 1988). They are *abstraction, focus, comprehensiveness, correctness, increment, standardisation* and *compatibility*. As shown in Figure 10, these recommended quality measures contribute to the three programming requirements of software quality, software reusability and software portability. Each of these measures are now critically discussed and used in the evaluation of object-oriented design for road traffic noise.

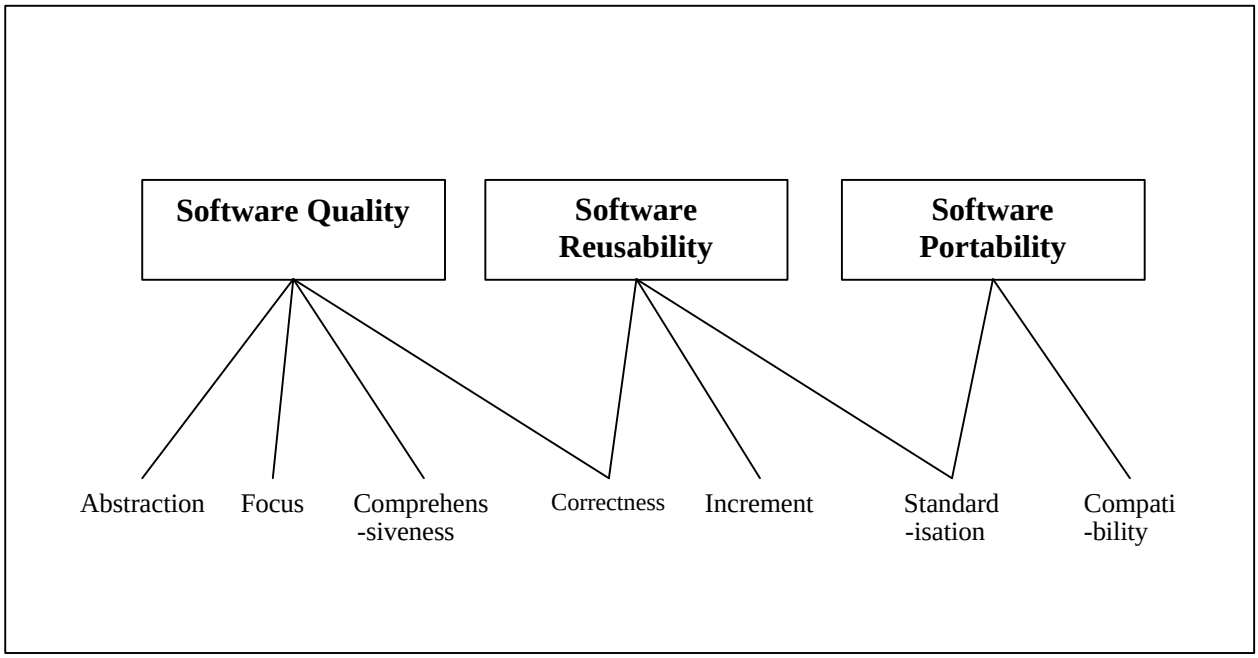

Figure 10: Evaluation Measures of Object-Oriented Design for Road Traffic Noise

In terms of the abstraction feature, each traffic noise class of objects is the realisation of an abstraction (ie. it is a complete model of the concept). For example, class Noise is a complete representation of a traffic noise model. Viewed from the perspective of software reusability, the abstraction feature in an object-oriented design makes it easier to identify the participating objects in specific problem such as road traffic noise. For example, the use of abstractions such as class Noise, class Noise_Segment, Noise Source, etc. to represent traffic noise problem helps to identify the hierarchical structure of a noise problem, a noise segment, a noise source, a noise barrier and a noise receiver.

The second feature is that each traffic noise class is narrowly focused. Each class represents only one concept. It is often expedient to consider expanding a class into a

bigger class to handle new, similar situations. The end result, though, is a component that is too large and too fragmented to be easily understood and the consequence is the software quality will be affected. By concentrating on one concept per class, each class as a resulting component is an easily understood unit that is more likely to be reused by another designer. More efficient applications can be built if the components used represent single concepts and code that supports the needed functionality in the component. As an example, instead of having a single class to handle every different type of traffic noise model and associated attributes, a total of ten classes have been developed to represent road traffic noise models.

The third feature is that each road traffic noise class is a comprehensive model of the concept represented by the module even if some of the facets are not relevant to a particular application. There are two reasons for comprehensiveness in the object-oriented design. First, a complete component is more likely to be reused. If services are missing from a component, a paraphrase would imply that these are precisely the services that will be needed by the next user. Second, a complete component is more economical to develop. It is widely accepted that the maintenance costs more per line than does original system development (McGregor and Sykes, 1992, p. 6). The cost of adding additional services to a module later is more than developing a complete implementation originally, particularly since changes to a module frequently affect components that already use it in its incomplete implementation.

The adopted strategy in developing object-oriented models is to place the emphasis of this feature on more generic abstractions and mechanisms such as point, line, matrix, vector and their associated linear algebra operations because these generic classes tends to be highly reused by others. For example, whenever there is a need to use spatial modelling tools, class Point and class Line should be considered as the first candidates.

Each road traffic noise class is aimed at correctly representing a model of the target concept. This correctness feature in object-oriented design of traffic noise is more easily attainable with object-oriented modules. The size and encapsulation of object-oriented modules makes testing and validation a more manageable task. In terms of software reusability, the correctness is an important factor in contributing to the confidence in using reusable software components.

The fifth feature in the object-oriented design is the support of incremental development. This is an important feature that takes advantage of the inheritance relationship. It is reuse in the sense that it requires the results of previous work in order to develop new products. This incremental reuse does not use existing components in an "as is" state; rather, it supports an evolutionary reuse. The incremental testing and documentation techniques reuse products from previous processes to reduce the effort needed to create some new product.

An important characteristic of the object-oriented approach which helps to maintain the incremental open structure of an object-oriented system is the polymorphism. With the polymorphism, the behaviour of any new specialised class which inherits from existing class can be flexibly handled through a flexible substitution policy. By using an appropriately abstract class as the formal parameter specification, the modeller can create a broad range of classes whose instances serve as the actual parameter to that method. The method to which these classes are parameters can be reused over a wider

range of possible parametric values. This increases the reuse of the method and of the class to which it belongs. This is the case for adding new methodology for traffic noise modelling such as the STAMINA model from the USA. Instead of rewriting the code for representing the STAMINA model, a new class can be added to the framework. A new class can inherit existing class and override the specific method to suit the STAMINA methodology. All other classes in the traffic noise framework are unchanged in providing support.

The sixth feature of object-oriented design is standardisation. It is aimed at capturing all the basic standard knowledge in the road traffic noise modelling process at different levels of interests in transport planning (spot level and network level). The basic input, output and modelling functions offered by road traffic noise models are to some extent becoming standard.

The final feature of the object-oriented design is the support of software compatibility. Graphical user interface (GUI) is the major concern in the compatibility of a software system. Different software systems implement different graphical user interfaces and they cannot communicate with each other. GUI is normally built on a certain window system. Microsoft Windows developed by Microsoft Corporation is becoming a standard for micro computers whereas X windows developed at MIT by many computer vendors and MIT is also getting very popular on UNIX-based work-stations. To deal with different windows systems, the adopted strategy is to externalise the GUI functionality. However, it does not mean that the GUI functionality is not considered in the development of object-oriented models. In fact object-oriented design provides maximum support to GUI development through the flexibility in design of the system (with a set of objects and messages sent among them) and class interface. This design will make object-oriented models applicable to a wide range of computers with different user interface designs.

Conclusion

The research reported in this paper identified two key issues in the demand for incorporating road traffic noise models in land-use/transport and environmental models. The data integration issue was discussed to support the use of GIS as a data integration platform. The model representation issue was raised from the need for a flexible representation of road traffic noise models covering from spot level to network level. In searching for a suitable representation/programming approach for road traffic noise modelling, different programming approaches were reviewed. With particular reference to the road traffic noise problem, the representation schemes from two approaches were compared: object-oriented and conventional approaches. The object-oriented approach was selected as a candidate for road traffic noise investigation due to its support to software flexibility and reusability. A complete object-oriented design was established to serve as a framework for representing all 12 possible configurations of road traffic noise problems. A worked example was used to demonstrate the implementation of an object-oriented design. The object-oriented design of road traffic noise has followed quite closely to the currently recommended quality measures for general software system.

One important finding from this research is the representation power of OOA in structuring road traffic noise. Software flexibility and software reusability are the two key features that add to the practicality of applying OOA in traffic noise modelling.

In concluding this paper, another important issue is raised. Most land-use/transport planning software packages developed have been considered as "one-off" applications. In other words, these packages are designed with a fixed analytical structure, fixed number and types of functions used, and leave only limited room for future modification or extension (eg. to include a new technique or enhance an existing technique). Consequently, these packages do not cope with the demand for improvement or modification in a cost effective way; in most cases new programs have to be written. Amongst those packages that exist many basic modelling functions are commonly used. For example, matrix manipulation functions represent the most commonly used tool in any transport software package. If software reusability is utilised, the duplication in building common software components can be minimised or avoided. There have not been any reports on a published library of models or functions being shared or re-used in the transport modelling area. This type of library is useful in terms of the software reuse of common software components to save the cost involved in the development of software for transport system analysis. This observation leads us to rethink the interaction between transport software developers and transport software users. The question is which side (developer or user) should commit more effort to this problem. Should the developer make transport software more flexible or the transport planner/engineer be more competent in transport computer programming? The challenge is how to develop a modelling/programming environment so that it can be used to help the transport modeller in representing a transport model to cope with an ever-changing real world problem. The object-oriented approach should be considered as a strong candidate.

Acknowledgement

The research and development reported in this paper is supported under the Australian Research Council Research Centres program.

References

Black, J, Trinder, J., Masters, E., Vandebona, U., Ton, T., Morrison, B. and Tudge, R. (1996) Spatial Decision Support System for Transport Planning, in D. Hensher, J. King and T.H.Oum (Eds.), Proceeding of the 7th World Conference on Transport Research, Sydney, Australia (Elsevier: Oxford), Volume 3, pp.71-83.

Booch, G. (1991) Object Oriented Design with Applications The Benjamin/Cummings Publishing Company: Sydney.

ESRI (1992) ARC-INFO Version 6 User's Guide, Environmental Systems Research Institute, Inc.: Redlands.

König, R. and Langbein, R. (1993) Simulation of Traffic Management Strategies, paper prepared for the 26th International Symposium on Automobile Technology and Automation, Aachen, Germany, September 13-17.

McGregor J.D. and Sykes, D.A. (1992) Object-Oriented Software Development, Van Nostrand Reinhold: New York.

McGurrin M.F. and Wang, T.R. (1991) An Object-Oriented Traffic Simulation with IVHS Application, Proceedings of Route Guidance and Vehicle Navigation Systems, pp.551-561.

Meyer, B. (1988) Object-oriented Software Construction, Prentice Hall: Sydney.

Ton, T.T. and Black, J.A (1993) An Object-Oriented Approach for Implementing an Integrated Four-Step Transport Planning Model, in R.E.Klosterman, and S.P. French (Eds.), Proceedings of the Third International Conference on Computers in Urban Planning and Urban Management, Atlanta, Georgia, USA, July 23-25, Vol. 2, pp.471-492.

Ton, T.T. (1995) An Investigation of the Analytical Capability of Object-Oriented Programming in Transport Modelling, unpublished PhD thesis, The University of New South Wales, School of Civil Engineering, Kensington.

UK Department of the Environment (1975) Calculation of Road Traffic Noise, HMSO: London.

UK Department of Transport (1988) Calculation of Road Traffic Noise, HMSO: London.

INSTITUTE OF TRANSPORT STUDIES

The Australian Key Centre
in Transport Management

The University of Sydney
and Monash University