TECHNICAL REPORT

ISO/TR 14813-4

First edition
2000-12-15

Transport information and control systems — Reference model architecture(s) for the TICS sector —

Part 4:
Reference model tutorial

Systèmes de commande et d'information des transports — Architecture(s) du modèle de référence du secteur TICS —

Partie 4: Manuel de modèle de référence

Reference number
ISO/TR 14813-4:2000(E)

Contents

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The main task of technical committees is to prepare International Standards, but in exceptional circumstances a technical committee may propose the publication of a Technical Report of one of the following types:

— type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;

— type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;

— type 3, when a technical committee has collected data of different kind from that which is normally published as an International Standard ("state of the art", for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

Technical Reports are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

Attention is drawn to the possibility that some of the elements of this part of ISO/TR 14813 may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO/TR 14813-4, which is a Technical Report of type 2, was prepared by Technical Committee ISO/TC 204, *Transport Information and Control Systems*.

This document is being issued in the Technical Report (type 2) series of publications (according to subclause G.3.2.2 of Part 1 of the ISO/IEC Directives, 1995) as a “prospective standard for provisional application” in the field of transport information and control systems because there is an urgent need for guidance on how standards in this field should be used to meet an identified need.

This document is not to be regarded as an “International Standard”. It is proposed for provisional application so that information and experience of its use in practice may be gathered. Comments on the content of this document should be sent to the ISO Central Secretariat.

A review of this Technical Report (type 2) will be carried out not later than three years after its publication with the options of: extension for another three years; conversion into an International Standard; or withdrawal.

ISO/TR 14813 consists of the following parts, under the general title Transport Information and Control Systems — TICS Reference Architecture:

— *Part 1: TICS Fundamental Services*: This document presents the definition of 32 TICS fundamental services that are the informational products or services or applications areas provided to a TICS user.

— *Part 2: Core TICS Reference Architecture*: This document describes an abstract object-oriented system architecture based on the TICS Fundamental Services.

— *Part 3: Example Elaboration*: This document refines the Core TICS Reference Architecture (Part 2) with some emphasis on traffic management.

— *Part 4: Reference Model Tutorial*: This document describes the basic terms, graphical representations and modelling views exploited in the object-oriented definition of the architecture development of Parts 2 and 3.

— *Part 5: Requirements for Architecture Description in TICS Standards*: Requirements for Architecture Description in TICS Standards: This document describes the terminology and form to be used when documenting or referencing aspects of architecture description in TICS standards.

— *Part 6: Data Presentation in ASN.1:* This document establishes the use of ASN.1 as the normal syntax notation to be used in standards for the TICS sector and a common message form for such ASN.1 based data elements.

Introduction

ISO/TC204/WG1 is a working group whose prime objectives are to provide services to ISO TC204 and its working groups. A specific mission of WG1 is to:

> "Provide ISO TC204, its working Groups, related bodies and those involved in the TICS sector, with a reference model of Conceptual Reference Architecture(s) that show the structure and inter-relationships of the sector ..."

It is expected that there may well be more than one single TICS Architecture approach to be considered and documented and that existing architecture approaches will have previously-produced documentation developed according to disparate standards and conventions.

It is also implicit in the work being undertaken by WG1, that working group members will require a clear, well-structured understanding of the work of the following participant groups:

- Other TC 204 Working Groups
- CEN TC 278 Working Groups
- Japanese initiatives
- European Road Transport and Traffic Telematics programs
- US Intelligent Transportation Systems program
- Australian initiatives
- Canadian Initiatives

Full documentation of all possible architectural approaches is obviously not feasible given the high level of resources required to carry this out. Indeed full documentation and description of all possible approaches is undesirable as an item for Standardisation.

A defined and consistent approach is however required to facilitate the specification of architecture requirements to enable a clear view to be developed and presented of the work of each participant group This document is one of a set of WG1 documents intended to respond to stated WG1 objectives regarding the production of a TICS Reference Architecture.

In order to document an architecture, graphical and textual components of a model are required. WG1 has adopted a methodology based on the Unified Modelling Language (UML) for documenting the TICS Reference Architecture. A tutorial on the UML is provided in ISO/TR 14813 Part 4. UML is a visual modelling language for building object-oriented and component-based systems. A commercially available Computer Aided Software Engineering (CASE) tool has been used by WG1 to document the Architecture. While the tool is a commercial product, UML is open and non-proprietary.

Transport information and control systems — Reference model architecture(s) for the TICS sector — Part 4: Reference model tutorial

1 Scope

The architecture of an information and control system merges hardware and software considerations into a coordinated and integrated system view. The system architecture is a high level abstraction, or model, of the system. A system architecture should embrace both today's applications and the applications that are expected in the future. Architecture begins with the definition of the conceptual services (e.g. Part 1 - TICS fundamental services). There are several identifiable stages of system architecture development.

- Reference architecture
- Logical architecture
- Physical architecture

The reference architecture is generic and non-prescriptive and captures the concepts of the system. A logical architecture elaborates the functions that will provide the conceptual behaviour, and in so doing it provides some detail about the modularity. A physical architecture is reached when the actual distribution of the system modules is defined, thus leading to important implications for communications.

This technical report develops a TICS Reference Architecture. The objective in defining a TICS Reference Architecture is to provide a concise reference point which is both educational and a framework for the standards process. The Reference Architecture will be used by the Working Groups to develop their own logical and physical architectures in a cohesive manner.

This Part introduces the model that is applied in developing the Reference Architecture in Parts 2 and 3. A tutorial on the application of the model is provided using examples from the TICS sector.

2 Modelling an architecture

In order to document an architecture, graphical and textual components of a model are required. A unified process and the Unified Modelling Language (UML)[1] developed by Ivar Jacobson, Grady Booch and James Rumbaugh addresses this requirement for the software industry. The result is a component-based process that is use-case driven, architecture-centric, iterative, and incremental.

In Parts 2 and 3 of this technical report the abstraction that is the TICS Reference Architecture is described in four views of the Unified Modelling Language:

1. Use Case diagram
2. Class diagram
3. Package diagram
4. Sequence (Interaction) diagram

The UML views have been developed to support methodologies for building systems with the object model (e.g. systems whose software is programmed in object-oriented languages). Therefore it is necessary to begin this tutorial by introducing some basic concepts of object-oriented (OO) modelling.

Although ISO/TR14813-5 states that architectures may be developed using either OO or process-oriented methods, WG1 has chosen to use the OO method.

3 Why object-oriented?

The Transport Information and Control Systems (TICS) Reference Architecture is being developed by ISO/TC204 Working Group 1 (WG1) with the following objectives.

- to be a concise statement of what TICS is, and thus help develop consensus within TC204 and wider circles
- to act as a framework for the standards process in TC204
- to serve an educational purpose and lend itself to promulgation in media such as the Internet.

A reference architecture is non-prescriptive of technology and deployment organisation.

In the future, WG1 expects a number of standards developments such as data dictionaries, model sub-system architectures, and message standards to update the reference architecture and to evolve it into various architectures.

Thus the choice of both model and process for the architecture development were important decisions. Current and future requirements include:

- information models
- functional models
- dynamic models
- implementation models

Older system development practices are disjoint in the way that all these requirements can be met. For example, functional and data flow models must be combined with a separate model for information, and these two types of model do not integrate conveniently. Object-oriented modelling accommodates these requirements harmoniously.

Modern computer based system engineering for software development automates the object-oriented model. The process also incorporates other important models. The same tools used for system engineering are suitable for the development of architectures.

By applying the object-oriented model and process for the Reference Architecture a highly effective base is established for future developments.

4 The Unified Modelling Language

The Unified Modelling Language is a visual modelling language for building object-oriented and component-based systems. The UML was recently developed by Rational Software Corporation and its partners and is the successor to the modelling languages found in the Booch, OOSE/Jacobson, OMT, and other system engineering methods. UML has been progressed through the Object Management Group (OMG) for adoption as an OMG standard.

The developers of the UML recognised that the choice of what model projections one creates has a profound influence upon how a problem is approached and how a solution is shaped. They maintain that

- Every complex system is best approached through a small set of nearly independent views of a model - no single view is sufficient.
- Every model can be expressed at different levels of fidelity.
- The best models are connected to reality.

Rational Software Corporation, as well as many others, are incorporating the UML into their development process and products, which cover disciplines such as business modelling, requirements management, analysis & design, programming, and testing.

WG1 has adopted a methodology based on UML for developing and documenting the TICS Reference Architecture and is using a commercially available Computer Aided Software Engineering (CASE) tool to do this. While a CASE tool is a commercial tool, UML is open and non-proprietary.

Abstraction, which is to focus on relevant details while ignoring others, is the key to the development of a reference architecture. This determined the selection of four particular elements from the UML for the modelling task:-

1. Use Case
2. Class
3. Package
4. Sequence (Interaction)

These elements provide the multiple perspectives of the reference architecture.

- Use case diagrams define the architecture boundary, the external actors and the services provided.
- Class diagrams define the abstract elements that comprise the reference architecture. These elements underpin the dynamic model of the system.
- Package diagrams are the means by which model (architecture) elements can be grouped. Packages provide a hierarchical organisation as well as a network of package references.
- Sequence (Interaction) diagrams describe how the implied objects of the system cooperate to provide the services defined in the use case.

Each element view is presented as a diagram. The underlying model integrates these views into a self-consistent reference architecture. The modelling elements are explained in more detail in the following sections.

It is expected that some of the other types of diagrams defined in the UML (e.g. implementation diagrams) may be used by Working Groups to develop more detailed architectures.

5 Object-oriented modelling elements: class and object

The main purpose of modelling is to prepare generic descriptions that describe many specific particular items. In object-oriented modelling the most important generic description is called a **class** and a specific particular item belonging to a class is called an **object**.

Table 1 lists three particular objects (instances of a class called Intersection). The corresponding real life situation is depicted in the schematic of Figure 1.

There are at least three conceptual levels involved in the example. The first conceptual level is the class. This is used in architecture and design. The second conceptual level involves the software objects belonging to the class (e.g. Intersection) which arise in the development of a system conforming to the architecture. The third conceptual level is that of the real world where there is roadway pavement corresponding to the higher level intersection concepts. The higher levels are abstractions that represent the real world objects for particular purposes such as system architecture and system implementation.

Table 1 — Example class and objects

Class	Software Objects	Real World Objects
Intersection {operations & attributes}	Object1{ data}	Main St & Route 401
	Object2{ data}	Main St & Washington St
	Object3{ data}	Main St & Route 20

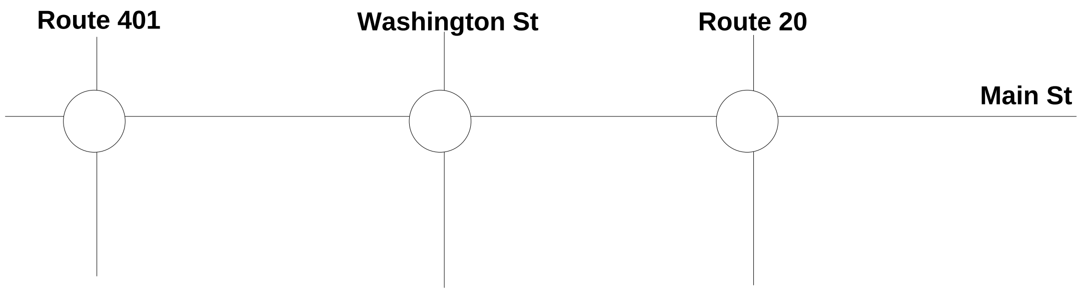


Figure 1 — A map of the real world corresponding to the three Intersection objects

In the above example the class might be defined so that information about real world intersections could be stored and updated in a system database. The reader shall see later that classes (and their implied objects) are often invented for other purposes, such as control and system interfaces.

Only classes are presented in the Reference Architecture, however the implied objects will often be referenced in the discussion of the architecture. A more complete definition of class is now given.

A **class** is the descriptor for a set of objects having similar structure, behaviour, and relationships. Class definitions consist of attributes and operations.

Attributes are the elements used to record the state of an object (e.g. the equipment deployed at an Intersection, or the traffic demand at an Intersection, or the current control phase). State changes of an object are recorded by changing the attribute values. **Static attributes** are those that apply collectively to the entire class (e.g. a count of Intersections). In Table 1 the state of the software objects is referred to as data.

An **operation** is an action that a software object performs (e.g. Change Phase at an Intersection in order to allow a different traffic movement) when it receives a particular stimulus called a **message** (e.g. a request to change phase). The collection of defined operations specifies the behaviour of a class (i.e. for all its objects). Classes may also have **static operations** (e.g. list the Intersections on a given route).

The relationships that are recorded about objects are called **associations** (e.g. each Intersection object is associated with a set of traffic Movements, and a Movement is associated with a set of Manoeuvres which can occur simultaneously, i.e. in the same traffic control phase). The various types of association are defined in a later section.

6 Abstraction

Unlike any other model, the class-object model can be applied in every stage of the development process, from architecture to system implementation. The key is the use of abstraction.

Because a reference architecture is concise and it originates at the very early stages of system development, it must use abstract classes, that is classes which reflect relevant details while ignoring others.

On the other hand, in object-oriented software design and implementation, the class element provides concrete specifications for the attributes, operations and state transitions of the software objects that will materialise the system.

Abstraction in the class-object model is a powerful means by which the strong connection between architecture, implementation, and the reality found in objects is maintained.

Abstraction is also applied in the other model views of the architecture. However it is only the elements of the class-object model which are materialised in the system implementation.

7 Model Views

Our objectives for conciseness etc. can only be met by using a graphical modelling language. We now describe the four UML model views and their diagram notation used in developing the reference architecture.

7.1 Use case diagrams

Use case diagrams are the first step in architecture development. They model the functional system requirements and scope. There are two primary components of a use case diagram, actor and use case.

An **actor** is a class external to the system. The relevant actors are those whose objects interact with the system. Thus an actor may be a role performed by a human or it may be a type of system. Although an actor is not necessarily a human it is always represented by a 'stick figure' in the use case diagrams, and labelled with its class name.

When an actor object interacts with the system, together they perform a behaviourally related transaction sequence. Each specific sequence is called a **scenario**. The abstraction of similar and closely connected scenarios is called a **use case**.

Just as the idea of classification is natural to the modelling process, so is the grouping together of strongly related use cases. This is the basis for forming a use case diagram. In the diagram a use case is represented by an ellipsis labelled with the name of the use case.

The natural way to begin an architecture specification is to identify some of the actors and the associated use cases. This is often referred to as developing the operational concept for the system. This process answers questions such as why is the system being developed and what must it do. (We might then specify some of the other types of diagrams before identifying all the actors and uses cases in iterative fashion.)

A **use case diagram** shows the relationships among the actors and the use cases. It is a graph consisting of actors and use cases, the latter enclosed by the system boundary. The system boundary separates the actors from the use cases and is shown as a dashed-line rectangle. The communication (participation) associations between the actors and the use cases, and the relationships among the use cases are shown by different types of arc in the graph.

The participation of an actor in a use case is shown by connecting the actor symbol to the use case symbol by a solid path. The actor is said to communicate with the use case.

A uses relationship between use cases is shown by a directed line from the use case drawing upon transactions outside itself to the use case being used. The line is labelled "uses". A uses relationship from use case A to use case B indicates that an instance of the use case A will also include the behaviour as specified by B.

The example use case diagram (see Figure 2) shows three actors and three use cases which could arise in the requirements of traffic management. (This is not a complete description of traffic management.)

The actors Vehicle, Traffic Operator and Parking interact with the use case called Traffic Control. Traffic Control uses two other use cases, Performance Evaluation and Performance Prediction. The relationships involving the latter are not fully developed in this example.

An important part of the use case model view is the accompanying free text description of the use cases and the actors. This documents in narrative form the logic of the associated transaction sequences. Other model views (e.g. sequence diagrams) are then developed in order to translate this documentation into a more formal notation.

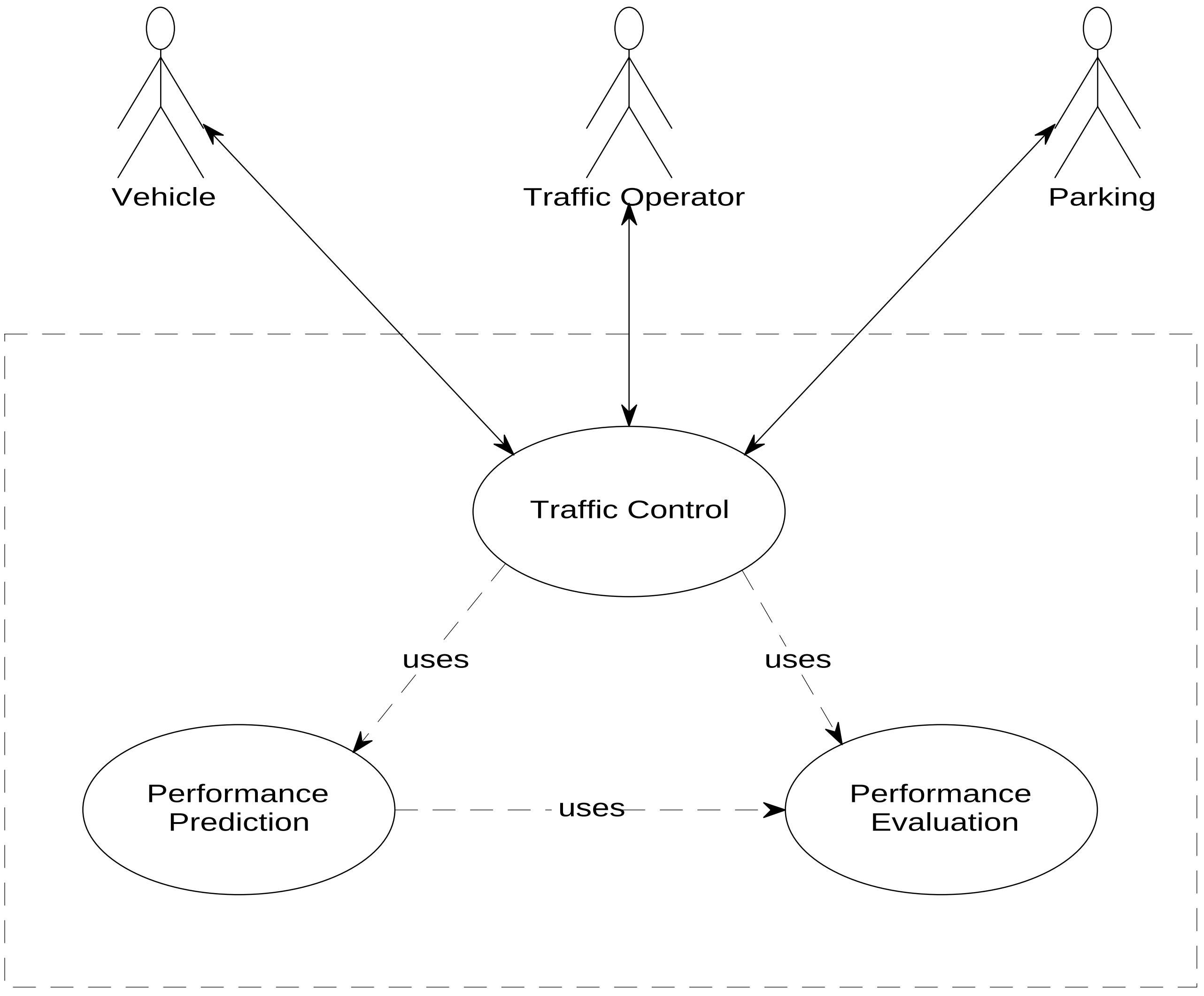


Figure 2 — Example use case diagram

7.2 Class diagrams

Classes are declared in class diagrams and they are used in most other diagrams.

Class diagrams show the static structure of the model, in particular, the things that exist (such as classes and their object sets), their internal structure, and their relationships to other things. Class diagrams do not show temporal information, although classes may contain attributes and operations that have or describe temporal behaviour.

A class is drawn as a solid-outline rectangle with up to three compartments separated by horizontal lines. The top “name” compartment holds the class name; the middle “list” compartment holds a list of attributes; the bottom “list” compartment holds a list of operations.

Either or both of the attribute and operation compartments may be suppressed in a class diagram. If a compartment is suppressed, no inference can be drawn about the presence or absence of elements in it.

The Reference Architecture does not define attributes because of its level of abstraction. This important element of the object-oriented model is mentioned for completeness and because attributes will arise when the Reference Architecture is elaborated by the Working Groups in conjunction with the development of data dictionaries and message standards.

An **operation** is shown as a text string that can be parsed into the properties (**signature**) of the operation model element. The default syntax is:

name (parameter-list)

where name is an identifier string;

where parameter-list is a comma-separated list of formal parameters.

In some cases the operation name is used without its parameter list.

In many class diagrams of the Reference Architecture both the operation and the attribute compartments are suppressed in order to avoid clutter, or because these elements are not yet defined. Operations occur explicitly in the sequence (interaction) diagrams that are described below.

The example in Figure 3 shows two class notations for a class named Public Transport Schedule. In the first only the name of the class is shown. In the second the name and the example operations are shown. Note there are no attributes or operation parameters. The example is too abstract to usefully identify that level of detail.

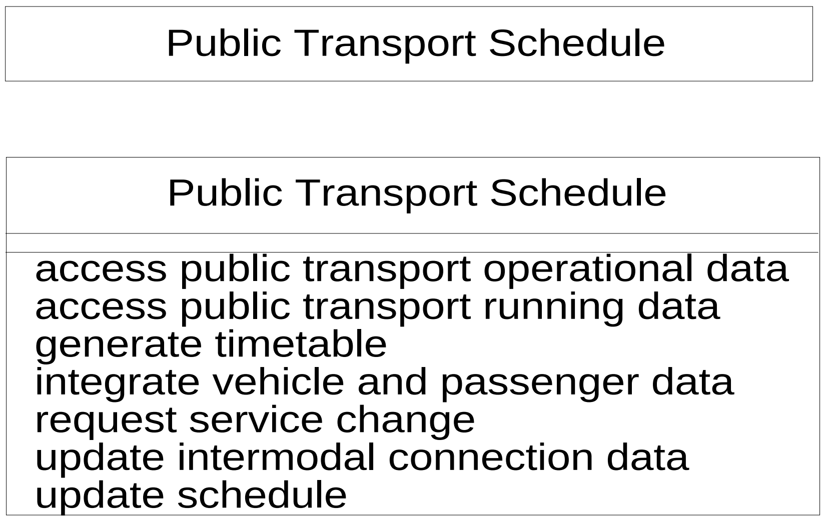

Public Transport Schedule
Public Transport Schedule
access public transport operational data
access public transport running data
generate timetable
integrate vehicle and passenger data
request service change
update intermodal connection data
update schedule

Figure 3 — Alternative notations for example class Public Transport Schedule

7.2.1 Associations

Classes can be associated in various ways. Depiction of these associations provides the structure evident on most class diagrams. These associations are now discussed.

Associations are relationships among classes. They are shown on class diagrams as solid lines connecting two class symbols. The lines may have a variety of adornments to show properties [e.g. the (optional) name of the association attached to the main part of the path].

Figure 4 illustrates the association named Controls between the class Incident and the class Incident Response. An instance of this association relates an Incident Response object with an Incident object.

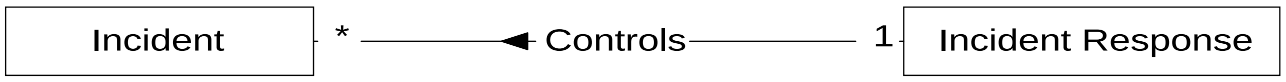


Figure 4 — Association named Controls relates Incident Response with Incident

The end of an association where it connects to a class is called an association role. The role is part of the association, not part of the class. Each binary association has two roles. Most of the interesting information about an association is attached to its roles. The following kinds of adornments may be attached to a role:

- role name: An optional name string near the end of the path which indicates the role played by the class attached to that end of the path.
- multiplicity: The number of objects which may be involved in a particular instance of that role. It may be unspecified.
- aggregation indicator: A hollow diamond is attached to the end of the path to indicate aggregation. The diamond is attached to the class that is the aggregate.

Figure 5 includes rolenames ("event" and "response") and multiplicity symbols (1 and *) for the association named Controls. At the “event” end of the association the asterisk means that an Incident Response object can be a “response” to zero or more Incident objects. At the “response” end the number 1 means that an Incident object has only one Incident Response “response” object.

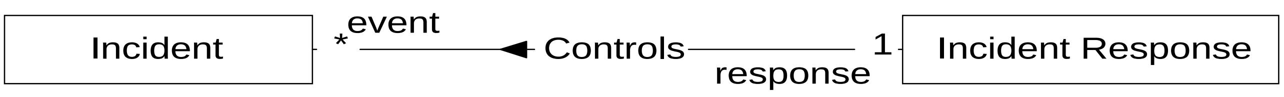


Figure 5 — Rolenames and multiplicity symbols for the Responses association

In some cases both ends of an association may be connected to the same class, but the two ends are distinct. Figure 6 shows the class Manoeuvre (i.e. a class whose objects each describe a traffic path which begins in one road element and ends in another road element). The association named “Conflicts with” is used to late Manoeuvre objects for which the traffic streams cannot occur simultaneously (e.g. objects whose paths cross).

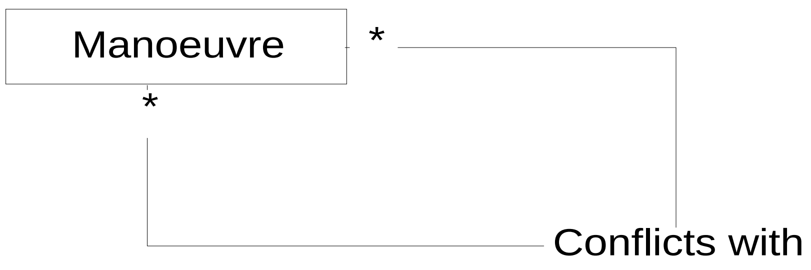


Figure 6 — Manoeuvre associations

The multiplicity * indicates that each Manoeuvre object conflicts with zero or more Manoeuvre objects. In the case of Manoeuvre objects at a controlled location the multiplicity would be a minimum of one other Manoeuvre object.

7.2.2 Special types of association

There are several types of association that need special note because of the richness of the model or their importance.

7.2.2.1 Aggregation

An example of an aggregation is shown in Figure 7. Each Movement object is comprised of one or more Manoeuvre objects. (In typical traffic control situations a Movement has at least two Manoeuvres, e.g. the flows in opposite directions along a two way road.)

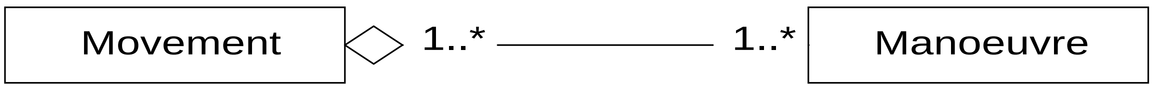


Figure 7 — A Movement is defined to be an aggregation of one or more Manoeuvres

7.2.2.2 Composition

Composition is a form of aggregation with strong ownership and coincident lifetime of the part with the whole.

7.2.2.3 Associations as classes

In some modelling situations it is desirable for an association to have attributes and operations. In this case the association is declared to be a class.

In CEN TC278 Geographic Data Files (GDF), a Manoeuvre is defined as an ordered sequence of a Road Element, a Junction and one or more Road Elements. A Road Element has a Junction as each end. A Junction represents the physical connection between its Road Elements (and any Ferry Connections). The direction along the first Road Element of a Manoeuvre corresponds to a Manoeuvre Set (our definition) which covers all possible subsequent paths from the initial Junction. Figure 8 is a representation of the Manoeuvre Set and the GDF Manoeuvre. The dashed line symbol indicates that the association (between Junction and Road Element) is being declared as the class connected (Manoeuvre Set).

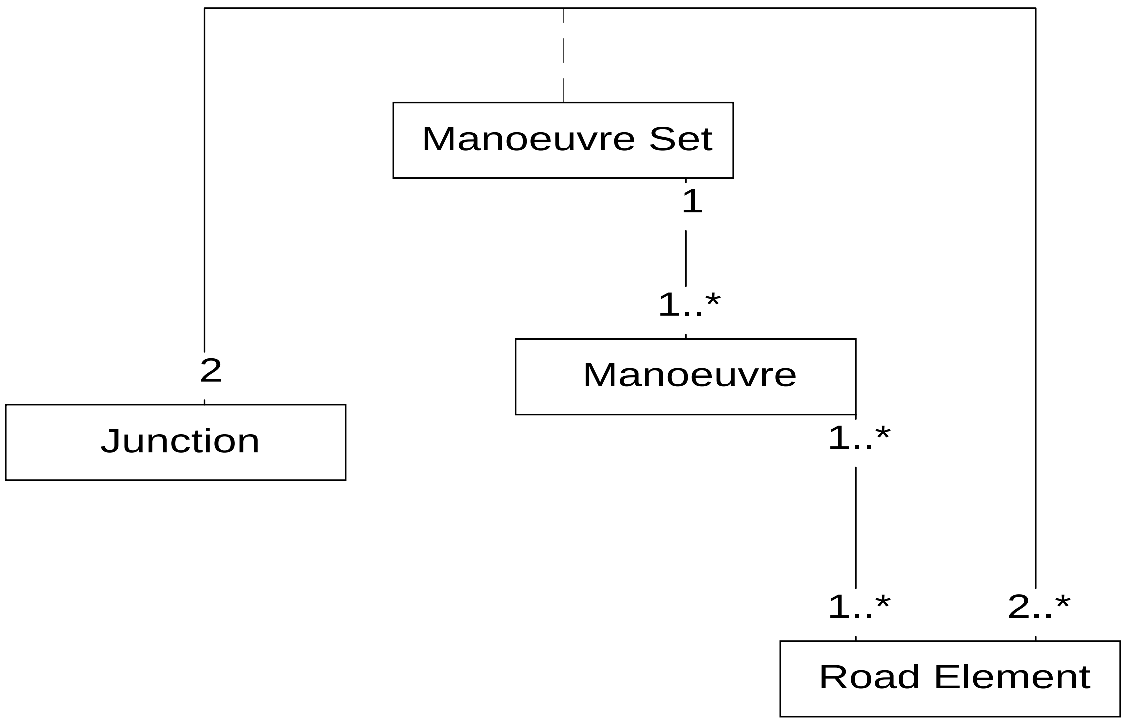


Figure 8 — Manoeuvre Set is an association class

7.2.2.4 Generalisation

Generalisation is the taxonomic relationship between a more general class and a more specific class that is fully consistent with the first class and that adds additional information.

Generalisation is shown as a solid-line path from the more specific class (the subclass) to the more general class (the superclass), with a large hollow triangle pointing at the more general class.

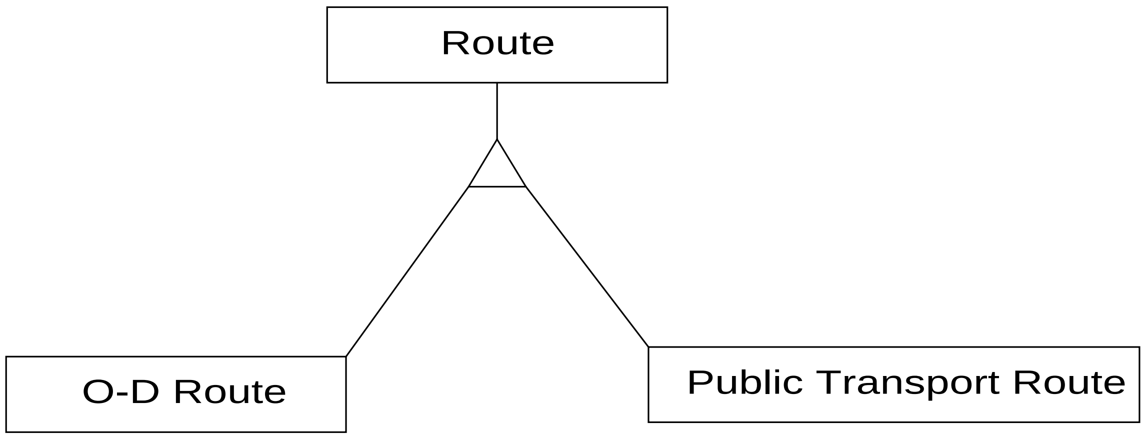


Figure 9 — Generalisation association between a superclass and two subclasses

A group of generalisation paths for a given superclass may be shown as a tree with a shared segment connecting to the superclass, and branching into multiple paths to each subclass. Figure 9 shows a superclass named Route which is specialised into two subclasses, Public Transport Route class and O-D Route class.

7.3 Package diagrams

Package diagrams are a grouping of model elements. Packages may be nested within other packages. Thus they are a means of avoiding information overload on any one diagram. Packages are another tool for abstraction. The entire system description can be thought of a single high level package with everything else in it.

In architecture a package is a convenient way of representing a grouping of closely associated classes. Thus a package can be used to represent a class diagram at a larger, less-detailed scale.

A package is shown graphically as a large rectangle with a tab attached to one corner. The name of the package is shown on the tab.

Relationships may be drawn between package symbols to denote relationships between at least some of the elements in the packages. In particular, a dependency relationship between packages implies that there exists one or more dependencies among the elements. A dependency is shown as a dashed arrow and indicates that a change to the target element may require a change to the source element in the dependency.

Figure 10 shows a possible set of high level packages comprising a TICS system and two of the many dependencies that might exist. This diagram defines a configuration for the architecture.

7.4 Sequence (Interaction) diagrams

Interaction modelling is used to specify the functionality in a use case in greater detail, and to portray how that functionality can be supported by the objects of the class model.

In performing the transactions of a use case scenario, there is always a sequence of messages between objects. The messages correspond to object interactions. In an interaction diagram this message sequence is generalised at the class level, showing class to class interactions.

Each inward message has a corresponding operation which it stimulates. The message carries the parameters for the operation which may in turn generate messages to be transmitted to other objects. Ultimately the message chain terminates and a response is generated which builds back towards the originator.

This sequence of message interactions is modelled in an interaction diagram. The diagram has two dimensions, the horizontal dimension represents classes, the vertical dimension represents time. Each class is represented by a labelled vertical line. There is no significance to the horizontal ordering of classes. Time proceeds down the page but only the downward sequence is important.

The notation also allows the message interaction to be described in a child sequence diagram. (This provides structure between use cases.) The horizontal line for that message terminates in an ellipse and is labeled with the name of the child sequence diagram.

Figure 11 shows the interactions for the use case named Performance Prediction. This use case is used by other use cases such as Traffic Control. The scenarios involved with this use case are centred on the Roadway Group object which predicts performance for its own network extent. In general this involves the following *sequence* of operations and interactions (thus sequence diagrams are also called interaction diagrams).

1. evaluate current performance within its network (object operation)

2. get data about all relevant Incidents within its network (interaction with Incident class)

3. get data from surrounding Roadway Groups which will impinge on performance in this network (static (class) operation)

4. do the prediction calculations (object operation)

5. record the predictions against the relevant (O-D Route) objects for reference by other use cases (interaction with O-D Route class)

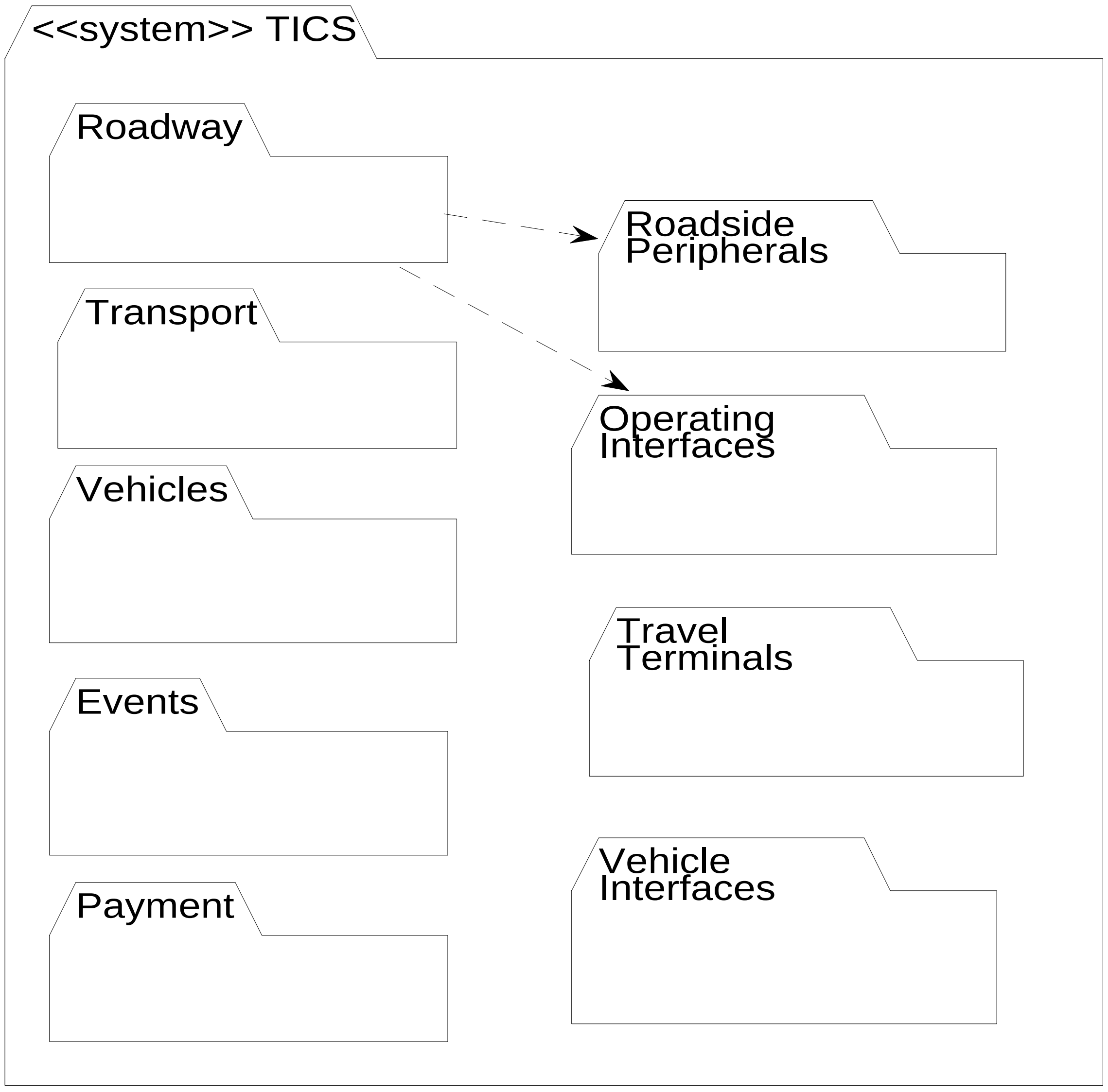


Figure 10 — Example package diagram for TICS

Each message is represented by a horizontal directed line labelled with a message identifier. For example, “access predicted incidents” is a message to the Incident class, and this class has a corresponding operation. The box on the vertical line represents the running of the operation in the object of the identified class. The return message from an object interaction is always implicit in interaction diagrams.

The flow of interaction messages will often be nested or non-sequential. The flow is described in structured English on the left hand side of the interaction diagram, using three types of statement:

- Selection - to describe a conditional statement (IF…THEN)
- Iteration - to describe a loop statement (DO…WHILE, etc.)
- Sequence to describe sequential process

In Figure 11 there is one iteration corresponding to periodic performance prediction.

The System Boundary (non-UML concept) represents the division between the objects in the use case and the actors, and is represented by a thick shaded vertical bar.

The Architectural Boundary (non-UML concept) divides the interface objects and the business objects and is represented by a vertical dashed line.

Figure 11 does not include any interface classes.

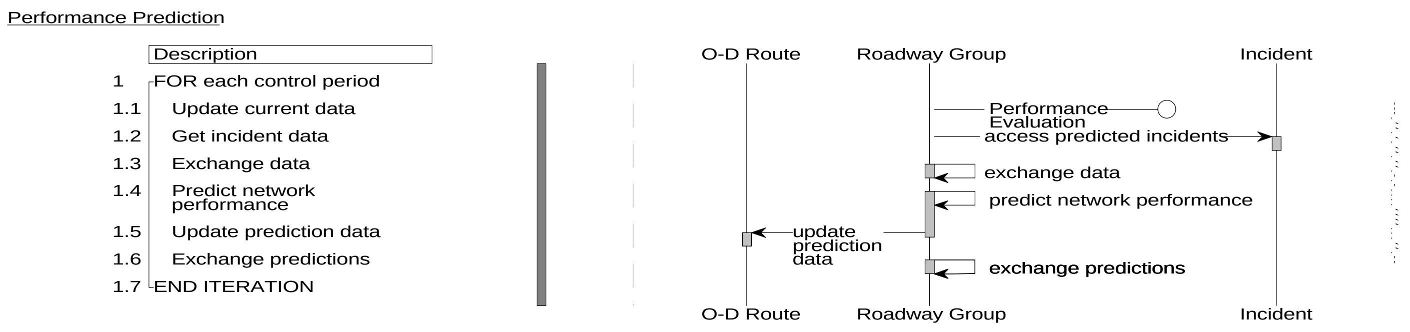


Figure 11 — Interaction diagram for a use case named Performance Prediction

8 Methodology

UML is a visual modelling language. A methodology is required to apply UML to developing an architecture. This section offers some insight into such a methodology. New terms and graphics introduced in this section are not part of the UML.

The development of an architecture for a complex system is never a sequential process. However there are clear relationships in any development methodology. Important relationships of a methodology can be identified by considering which UML elements and diagrams feed into others.

The methodology begins with a requirements gathering exercise for which the elements of use case diagrams are the main modelling aids. The use case diagrams are developed by considering the ‘why” and the ‘what’ of the rationale for the target system. Less formal requirements exercises may precede the generation of use case diagrams. ISO/TR 14813-1 documents the TICS Fundamental Services. These provide the basis for identifying actors and use case. The grouping of actors and use case linked by relationships allows the development of use case diagrams as indicated in Figure 12.

The remaining part of the methodology is highly iterative. The textual documentation of a use case is formalised in a sequence diagram. Thus there is a primary input from use case to sequence diagram. However a sequence diagram cannot be built without classes for which the necessary operations have been defined. Thus there is a primary input from class diagrams into sequence diagrams and it is necessary to define some classes before proceeding to any sequence diagram.

Another process which occurs continually from architecture to design is the replacement of an abstract set of elements with a more concrete set. This also exercises the iterative relationships shown in Figure 12. Package diagrams are a useful modelling view for identifying modularity and concisely representing all the classes which are present in a class diagram. A package may be identified before its component classes.

This covers the four views that are used in TICS architecture. The development of classes, which form the body of the architecture, is now discussed in more detail.

Invention is a key part of architecture development. In the methodology invention is centred on the definition of the classes and their operations. Reuse is an important concept in class definition. Thus classes are identified after consideration of all the use case which might apply. The relationships among use case are relevant and there is an iterative relationship in the development of class diagrams and use case diagrams.

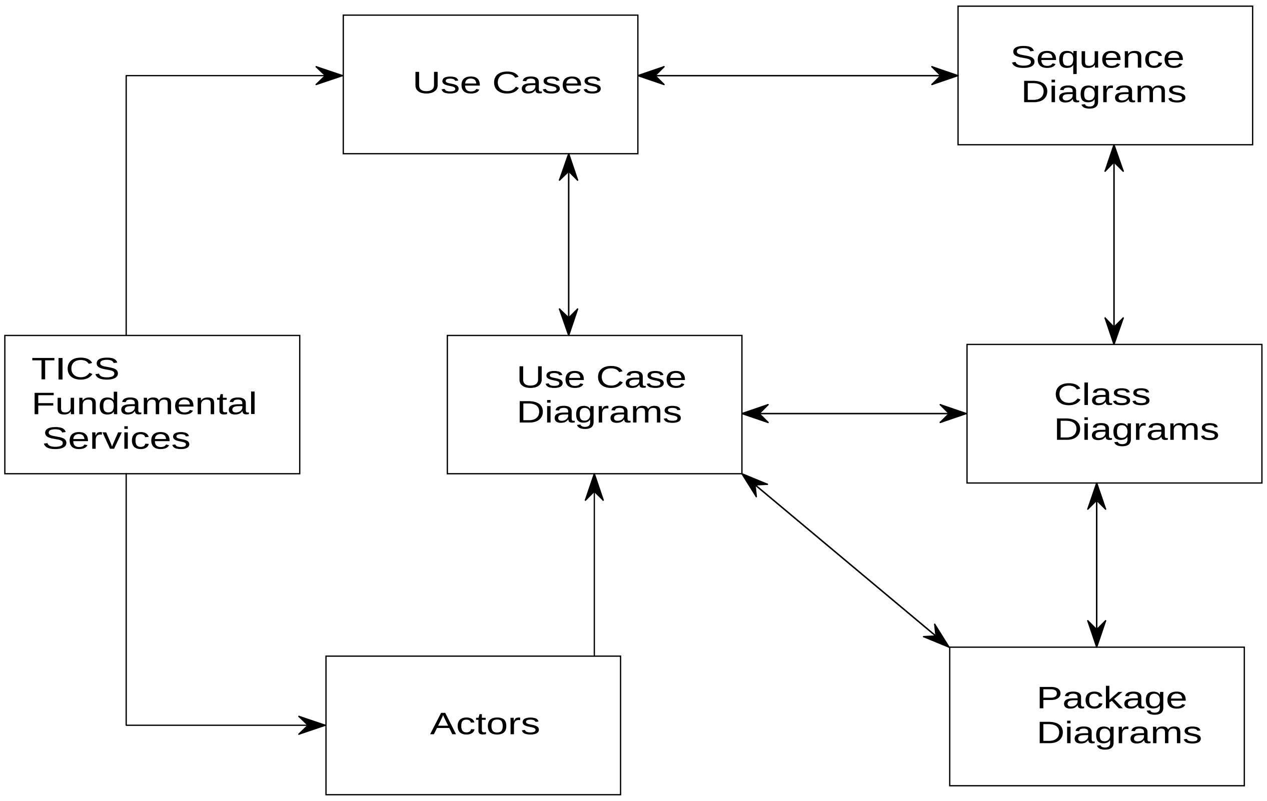


Figure 12 — Relationships of the development methodololgy

The methodology for the development of classes is based on a consideration of what is needed to provide the services of each use case. In object-oriented terms service implies invoking the operations of objects. Therefore classes are formulated under three guidelines listed below, all the time connecting to reality.

- information
- control
- interface

The example class diagram in Figure 13 is now explained. The diagram contains six classes. The associations describe each Roadway Group object as an aggregation of a Roadway Network object, and zero or more Local Control Group objects. A Roadway Group object is associated with one or more Control Plan objects and zero or more Parking objects. The asterisk at the end of the association with Parking means that for any particular Roadway Group object there can be no such association, or associations with one or more than one Parking object.

Information classes define objects which model information in the system which is held for a significant period of time, and which would typically survive one use case scenario. The operations of the information classes are primarily about creating, updating and accessing the data stored in their attributes.

For example, consider the use case named Traffic Control in Figure 2. Some of the information essential for this use case calls for an information class named Control Plan (Figure 13). The attributes of Control Plan embody the strategy to be applied or parameters to be used in the computation of “traffic control”. Sometimes these objects are generated through another use case (Transportation Planning Support). The objects reside in the system for a long time, being subject to ongoing maintenance through the interactions of the actor Transport Planner.

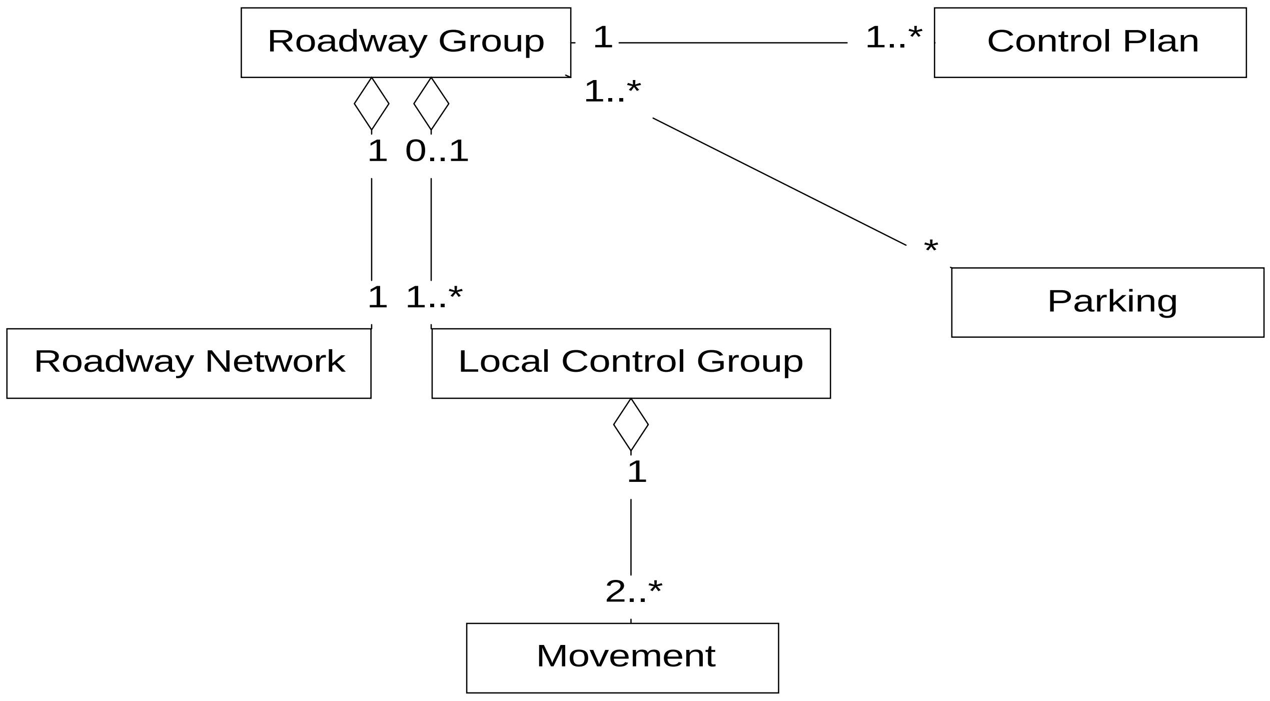


Figure 13 — Class diagram with traffic management connotations

For some actors a corresponding information class is required. These classes maintain information inside the TICS system about the actor objects, for example, the actor named Parking. The actor Parking is a class of external systems which operate autonomously, but which cooperate with TICS in use cases such as Traffic Control and Demand Management. There is a need for a TICS information class named Parking (Figure 13) in order that the some state of these actor objects can be maintained as continuously available information within the TICS system.

Referring again to Figure 13, traffic control requires a model of the traffic flow which is deeper than the individual elements of the roadway. So an information class called Roadway Network (which when analysed is an association between other classes such as Intersection and Link) is included.

The class Roadway Group is an aggregate of the relevant component classes. However Roadway Group is not just an information class. It is primarily a control class (see below).

Control classes are defined to model functionality that is not naturally tied to any other class. For example, a control class might have an operation, the execution of which might be invoked by an interface object, and it consists of interacting with several different information objects, doing some computation, and then returning the resulting data to the interface object and ultimately to an actor.

Consider the class Roadway Group of Figure 13. This class is defined so that the traffic on the corresponding real-life intersections and links (the Roadway Network) can be controlled by the system (e.g. Traffic Control, Incident Management and other use cases). There are numerous operations involved. An example of an interaction sequence based on the Roadway Group is:-

1. compute plan (Control Plan)

2. update control data (Roadway Network)

3. update control data (Parking)

Steps 2 and 3 would trigger other interactions which would result in delivery of control data to the actor or peripheral hardware of TICS.

Control classes have operations that are applied in one or more use cases. In the Reference Architecture, a control class will usually have an information role as well. Thus classes fall into a continuum between predominantly control and predominantly information.

The control class Roadway Group satisfies a good deal of its information functions through associations with other classes such as those depicted in the example of Figure 13.

Control classes give rise to most of the dynamic characteristics of an architecture. The performance of an operation of a control class often results in messages being sent to other control objects, thereby causing a sequence of operations involving numerous control and information objects. These sequences are identified in the interaction diagrams (see Section 7.4).

Some classes will identify important interfaces in the architecture (or public objects in a distributed object system) and the messages occurring in the interactions with these classes will serve the development of message standards.

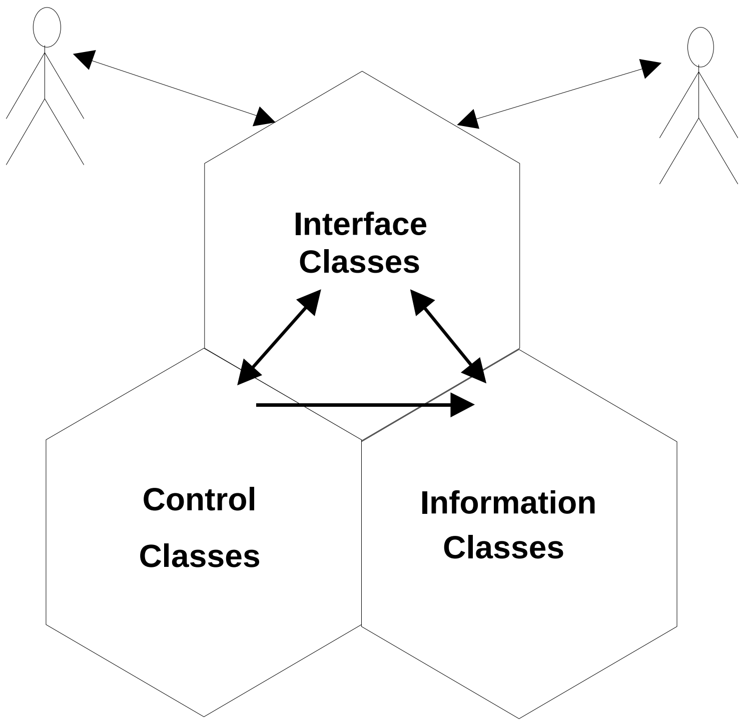


Figure 14 — A schematic view of the cooperation between classes in a high level architecture

Interface classes are defined to model the process of capturing information from, and presenting information to, an actor which is by definition outside the system boundary. Thus the architecture boundary is formed by interface classes for all the different operator and external system roles, and for all the situations where users can interact with the TICS system.

As illustrated in Figure 14 interface classes, control classes and information classes will all be involved in any actor transaction sequence (use case). Figure 14 shows the different interaction paths amongst classes in the system. Actors indirectly invoke operations of the control and information classes by invoking an operation of an interface class.

9 Summary

The abstraction of the TICS Reference Architecture is described in four model views

1. Use Case diagram
2. Class diagram
3. Package diagram
4. Sequence (Interaction) diagram

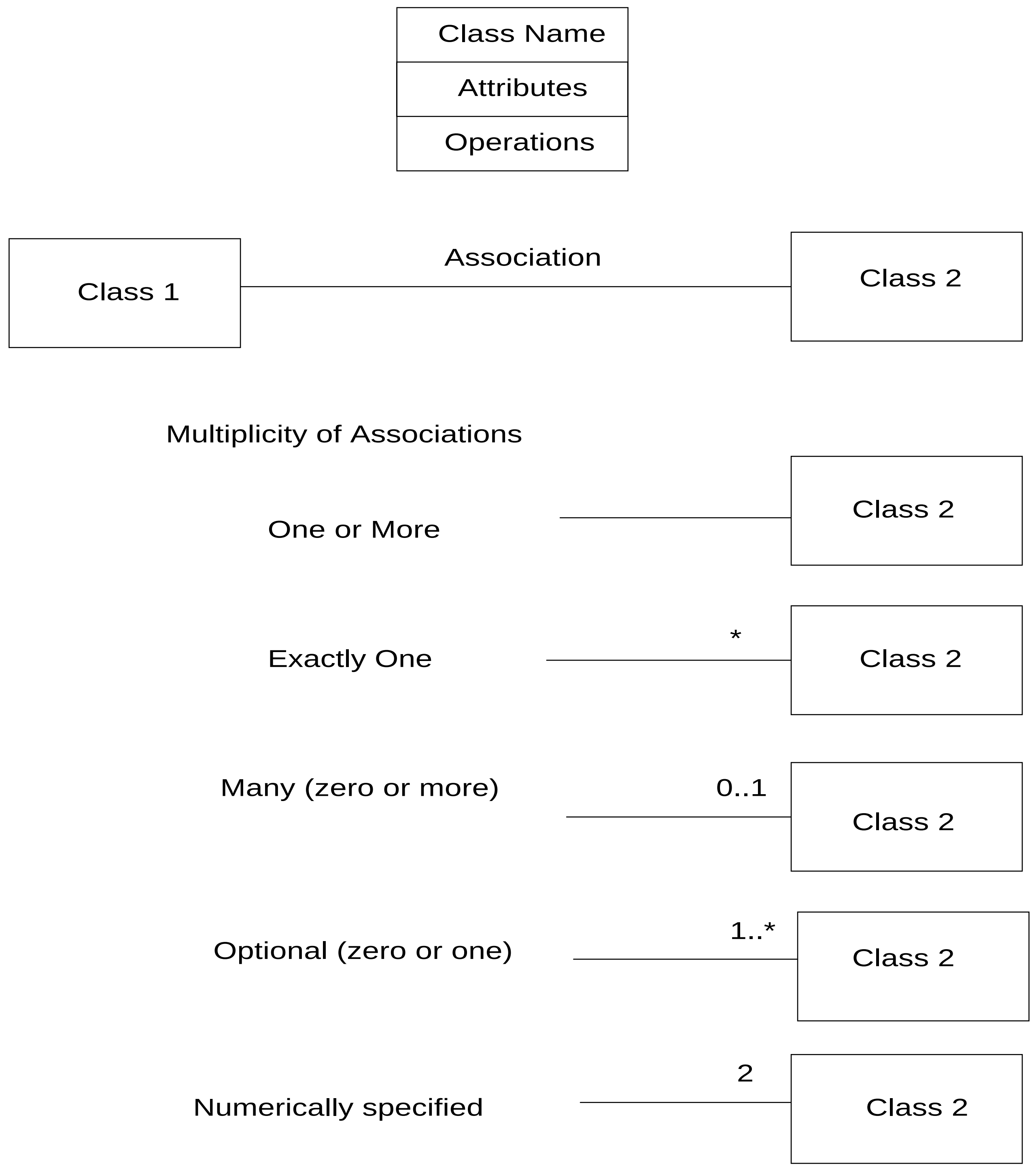
These elements provide the multiple perspectives of the reference architecture.

1. Use case diagrams define the architecture boundary, the external actors and the services provided.
2. Class diagrams define the abstract elements that comprise the reference architecture. These elements underpin the dynamic model of the system.
3. Package diagrams are the means by which model (architecture) elements can be grouped. Packages provide a hierarchical organisation as well as a network of package references.
4. Sequence (Interaction) diagrams describe how these objects cooperate to provide the services defined in the use case.

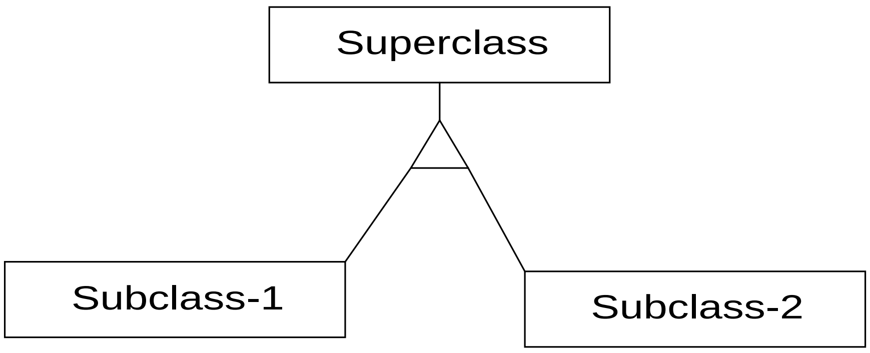
Each element view is presented as a diagram. The underlying model integrates these views into a self-consistent reference architecture.

Abstract class definitions are in keeping with the high level of the reference architecture. This means that the class definitions do not have attributes and the parameters of the operations may not be defined. Thus class definitions consist of the class name and operation names. These general characteristics are elaborated in descriptions recorded in the data dictionary of the CASE tool and distributed with the reference architecture documentation.

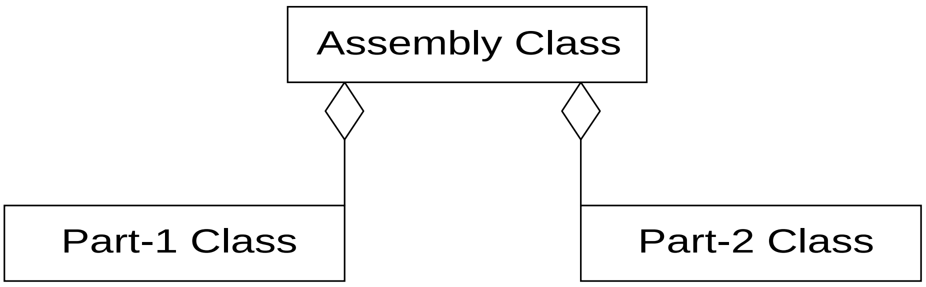
10 Legend of graphic symbols

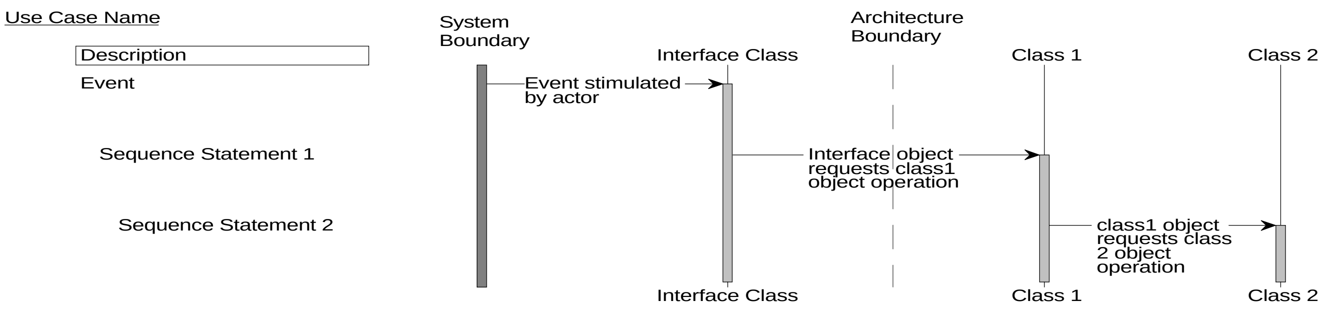


.
Generalisation (inheritance)



Aggregation





Bibliography

[1] The Unified Software Development Process, Ivar Jacobson, Grady Booch and James Rumbaugh, Addison-Wesley, 1999.

[2] The Unified Modeling Language User Guide, Booch G., Rumbaugh J., Jacobson I., Addison Wesley, 1999, ISBN 0201571684.

[3] Unified Modeling Language, UML Semantics, Version 1.1, 1 September 1997. Available on the Internet: <http://www.rational.com/uml>

NOTE At the time of the technical development of the TICS Reference Architecture, UML Version 1.1 was evolving. The software tool used in this effort provided one available presentation for UML diagrams. Since then UML Version 1.3 has been formalized and subsequently submitted to ISO under a Publicly Available Specification Procedure. UML Version 1.3 is being balloted as Draft International Standard ISO/IEC 19501.

ICS 03.220.01; 35.240.60

Price based on 21 pages