A Model for Mobile Spatial Services

Éamonn Linehan

A thesis submitted to the University of Dublin, Trinity College

in fulfillment of the requirements for the degree of

Doctor of Philosophy (Computer Science)

October 2008

Declaration

I, the undersigned, declare that this work has not previously been submitted to this or any other University, and that unless otherwise stated, it is entirely my own work. I agree that Trinity College Library may lend or copy this thesis upon request.

Éamonn Linehan

Dated: 25th March 2009

Acknowledgements

Éamonn Linehan

University of Dublin, Trinity College

October 2008

Summary

Mobile context-aware computing is a paradigm in which mobile devices have access to information, known as context, about the situation in which they are being used, and dynamically adapt application behaviour to support user tasks and mobility. A primary concern of mobile context-aware computing is awareness of the physical environment surrounding a user. This concern is addressed through spatial-awareness, where mobile applications employ knowledge about the physical location of real world objects to compute, for example, estimated journey times, routes between activities, perform proximity-based information retrieval, and render map-based interfaces. Geographic Information Systems (GIS) have traditionally performed these operations on static spatial data, vector-based geometry and attributes describing real world objects. However, spatial data and supporting services are confined to central servers, accessible to mobile devices in the form of a static graphic representation via wireless networks.

The range of services that spatially-aware applications on hand-held mobile devices can provide are limited by technical factors such as the inherent unreliability of wireless networks and the limited nature of mobile devices in terms of battery power, memory constraints and screen size. Because these limitations are common to all mobile applications, a generic model for spatial services is needed that is designed not to overburden the limited resources of mobile devices and is not dependent on continuous network connectivity.

This thesis proposes a model for spatial middleware featuring algorithms designed to minimise the processing time and power consumed on hand-held mobile devices, while providing uninterrupted access to common spatially-aware application services, such as rendering of geospatial information, real time generalisation of dynamic scenes, route generation, and visibility determination. The algorithms included in the model are designed to reduce the complexity of spatial data, thereby reducing the processing time and power consumed while manipulating it on mobile devices. These algorithms include a multiple representation database designed to approximate continuous scale adaptation with stepped levels of detail. Access to this data structure is facilitated by a hierarchical spatial index that uses minimum bounding boxes to approximate more complex spatial objects. The levels of detail are computed using algorithms that eliminate objects based on rendered size at particular scales and simplify geometry using shape and location preserving algorithms. Limited processing resources are further preserved by clipping geometry that extends beyond the viewport of the device to avoid computing projections and rendering coordinates that will not be seen. A graph-based topological representation of spatial data is searched using pluggable graph traversal algorithms with configurable cost functions to provide navigation capabilities.

The increased accessibility of spatial information offered by the model allows for the development of innovative services such as visibility determination, which expands mobile context-aware computing's environmental awareness beyond the physical location of real world objects to include the visibility of those objects. Spatial objects within the applications' viewport are filtered based on a field of view determined by compass direction and the human visual system. The resulting objects are then stored in a depth buffer on which a variation of occlusion culling is performed on their bounding volumes to reduce the set of geometry in the buffer to a possibly visible set (PVS).

Spatial services are incorporated into a generic framework, supporting the development of spatially-aware applications. Performance evaluations demonstrate that spatial services such as map rendering, generalisation, route generation and visibility determination can be provided locally on mobile devices. In addition, empirical experiments demonstrate that the model for spatial middleware presented in this thesis is is more energy efficient than existing server-based approaches. The reusability and extensibility of the framework to support the development of a range of mobile, spatially-aware applications is evaluated through the development of a case study application.

Publications Related to this Ph.D.

Éamonn Linehan, Shiu Lun Tsang, and Siobhán Clarke. Supporting Context-Awareness: A Taxonomic Review. Technical Report, Department of Computer Science, Trinity College Dublin, TCD-CS-2008-37, `http://www.cs.tcd.ie/publications/tech-reports/reports.08/TCD-CS-2008-37.pdf`.

Cormac Driver, **Éamonn Linehan**, Mike Spence, Shiu Lun Tsang, Laura Chan and Siobhán Clarke. Facilitating Dynamic Schedules for Healthcare Professionals. In *Proceedings of the 1st International Conference on Pervasive Computing Technologies for Healthcare*, Innsbruck, Austria, 2006. IEEE.

Cormac Driver, **Éamonn Linehan** and Siobhán Clarke. A Framework for Mobile, Context-Aware Trails-based Applications: Experiences with an Application-led Approach. In *Workshop 1 - "What Makes for Good Application-led Research in Ubiquitous Computing?", 3rd International Conference on Pervasive Computing (PERVASIVE 2005)*, Munich, Germany, 2005.

Éamonn Linehan, Cormac Driver and Siobhán Clarke. Route Generation for Adaptable Trails-based Applications. In *3rd UK-UbiNet Workshop*, University of Bath, UK, 2005.

Contents

List of Tables

List of Figures

List of Listings

Chapter 1

Introduction

This thesis investigates architectural models for spatially-aware middleware, including the design of a set of algorithms that provide reusable lightweight variants of spatial services on hand-held mobile devices. The provision of such services on mobile devices is limited by technical factors, such as periodic disconnection from wireless networks, the inherent unreliability of wireless networks and the limited nature of mobile devices in terms of battery power, memory constraints and screen size. The thesis describes algorithms designed to minimise the processing time and power consumed on mobile devices while providing uninterrupted access to spatially-aware application services. This chapter provides the background and motivation to this work, introduces an architectural model for spatial middleware services and concludes with the contributions and the layout of the thesis.

1.1 Background

This section describes the technologies involved in mobile spatially-aware computing (Figure 1.1). In particular, the characteristics and limitations of mobile devices and wireless networks are presented. A description of the nature of spatial information is provided, including a presentation of its importance to mobile computing, its structure and the common operations performed on it.

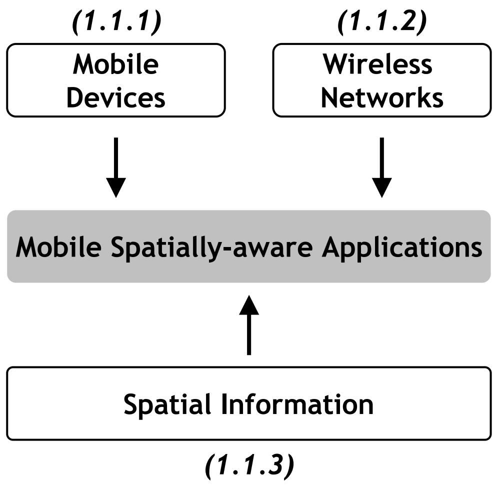


Figure 1.1: Background technologies

1.1.1 Mobile Devices

Hand-held mobile computers are small, portable devices that allow the user to move away from the traditional desktop environment, while retaining the ability to undertake computing tasks. Hand-held mobile computing devices include smart phones, pocket PCs and personal digital assistants (PDAs). Laptops and ultra mobile PCs can be considered a different class of mobile device and are explicitly excluded from consideration as they exhibit different characteristics to hand-held devices, demanding their own set of requirements and approaches.

Hand-held mobile computing devices differ from their desktop counterparts in terms of form factor, architecture, failure semantics and usage patterns [198]. The form factor of hand-held mobile devices is defined by the need to fit comfortably into one's pocket. Devices are intended either to be operated by one hand or to be held in one hand and operated with another (most PDAs fall into this category). The most significant architectural differences between mobile devices and desktop computers is their use of memory and integrated sensors. On a desktop computer, RAM and long-term storage space are separated, whereas, on mobile devices, RAM is often used both as working memory and long-term storage, limiting its availability to applications.

Mobile devices are increasingly being equipped with sensors that can be used to acquire information from the environment. For example, tilt sensors have been used to enable one-handed operations and to automatically adapt display orientation (landscape/portrait) to device orientation [193, 205]. Positioning sensors, the most common type of which is GPS, [66] are being used to determine location. Examples of other positioning systems include infrared, Bluetooth, ultrasonic beaconing [232, 233], wireless network fingerprinting [31] and network positioning based on cellular base station time of flight measurements. Other sensors commonly available on mobile devices include microphones, cameras, touch, and light sensors.

Mobile devices also have different failure semantics than their desktop counterparts. Unexpected failures are common. For example, communications failures occur often, users can remove the battery midstream, or the operating system may shut down applications when it is running low on resources.

Finally, the usage patterns of mobile devices are very different to desktop computers. Hand-held mobile devices and their applications are often left running ready to provide instant services to their users. They are used in crowded, noisy spaces, often in daylight conditions to perform activities that are focused and of short duration. For example, making a phone call, checking an appointment schedule, or reading instant messages are all done using frequent but short-duration sessions.

The mobile nature of these devices has led to form factor, architecture, failure semantics and usage patterns that place significant stress on the limited available resources in terms of battery, computation, memory, screen, input mechanisms and communication capabilities, as follows:

- *Battery* - Mobile devices rely on a finite energy source that is a major constraint in mobile environments. Although battery technology has improved over the years, it has not kept up with the computational demands of mobile systems [202]. To maximise the amount of work the user can do before the battery becomes discharged, hardware and software designers have to optimise energy consumption to avoid either heavier battery packs or short durations between battery recharges [41]. To be effective, mobile devices need to be able to operate untethered for long periods of time (typically devices are not connected to wired power or communication sources more than once a day).

- *Computation* - The design of CPUs for mobile devices is heavily energy constrained, resulting in CPU speeds of mobile devices being typically an order of magnitude less than desktop platforms (100-600MHz). Today's mobile devices do not feature graphics hardware[1], which means that the main processor is responsible for rendering 1.3user interfaces as well as performing all other computation. In addition, mobile devices often serve other critical purposes while running foreground applications that must remain available for users at all times. If a mobile phone ceases to work as a phone due to an application consuming all CPU resources or blocking the user interface, the user experience will be severely degraded.

- *Memory* - Many current devices have 64-128MB of program memory available [11]. However, applications using large chunks of memory (more than 4 MB) load slowly because of their volume, slowing down the entire device [225]. Mobile devices do not use page files to swap out memory not being used, due to their lack of disk drives. Instead, mobile devices often feature flash storage devices in the form of removable memory cards. However, accessing flash storage is slower than traditional RAM (particularly write operation latency) making it most suitable for storing static read-only data (for example, MP3 players).

- *Screen* - A social requirement for mobile devices to remain pocket-sized imposes a constraint on the maximum physical size of the screen. In addition, mobile displays also commonly have lower resolution, fewer colors, and a different width/height aspect ratio than desktop computers.

- *Input* - The small form factor of mobile devices has resulted in different input devices than desktop computers. Devices are often designed for one-handed use and commonly feature buttons, joysticks or touch screens displays.

- *Communication* - Current devices are typically equipped with several wireless network interfaces, such as 802.11, cellular (including 3G), and Bluetooth. A discussion of the specific limitations of these wireless network interfaces follows (Section 1.1.2).

These limitations pose significant design challenges to the developers of mobile, spatially-aware applications [202]. This thesis describes an architectural model for spatial midd-

[1]There is a small but rapidly growing number of UMPCs, PDAs and smart phones that have hardware 2D or 3D acceleration supported by Java ME, OpenGL ES or Direct 3D. A list of such devices is maintained by the Mobile Data Visualization Lab, University of California (`http://mobile.sdsc.edu/devices.html`).

leware services that does not overburden the limited resources of mobile devices and is not dependent on continuous network connectivity. These services are provided as a set of algorithms, designed with the form factor, architecture, failure semantics and usage patterns of mobile devices in mind.

1.1.2 Wireless Networks

The primary resource requirement of a mobile device, when it is working as part of a distributed system, is its network connection, which is usually some form of wireless connection [80]. Wireless networks are highly variable in performance, reliability and coverage. When used by devices that are moving, unanticipated and possibly prolonged disconnections can be caused by physical obstructions or lack of sufficient network infrastructure. For example, some buildings offer reliable, high-bandwidth wireless connectivity but thick concrete walls can block signals resulting in areas of the building where there is no signal. In addition, today's wireless networks typically operate on frequencies of 1GHz or more and signal power levels that result in a range limitation of approximately 100 meters. Outdoors, a mobile client may have to rely on cell-based low-bandwidth wireless networks, limiting throughput.

Mobile devices are sometimes organised in an ad-hoc and peer-to-peer manner. These networks of mobile devices are known as *Mobile Ad Hoc Networks* (MANETs). MANETs have no fixed infrastructure or centralised administration but are, instead, self-configuring. Nodes will usually send out their own requests, forward other nodes' requests, and respond to other nodes' requests during their participation in the network. MANETs cannot always provide access to centralised servers limiting mobile applications ability to access infrastructure-based services.

1.1.3 Spatial Information

Mobile devices are portable enough to be with us all the time, have wireless communication capabilities and are increasingly making use of integrated sensors and positioning technologies to gather information on the environment in which they are being used. Egenhofer termed this class of mobile device a, "spatial information appliance," as it is envisioned that such devices will open spatial analysis to day-to-day use by providing specialised spatially-aware applications tailored to particular tasks [65].

The usage patterns of spatial information appliances are such that the user's attention is often focused on other tasks, requiring unobtrusive interfaces that autonomously determine the information that is currently relevant to the user [183]. To provide such interfaces, it is necessary for mobile applications to be able to discover and take advantage of contextual information (such as user location, time of day, nearby people and computing devices, and user activity) [203]. Spatial information is an important component of such context-aware mobile applications for a number of reasons:

- Spatial information provides knowledge about the user's environment beyond what can be gathered from sensors directly. Sensors detect useful information such as position, orientation, and light levels but do not capture the kind of knowledge that is encapsulated in digital maps [36, 169]. For example, sensors cannot automatically gather information on the layout of buildings or cities in order to provide users with directions.

- Spatial information models facilitate the construction of a view of the world that drives application behaviour [152]. This model of the world is exploited by mobile applications [129, 51, 68, 223] to judge distance [184, 134, 178], determine visibility [84, 87, 132], produce routes [13, 165, 121], and provide other useful services to mobile users.

- Spatial information can serve as a shared metaphor between the system and the user in the same way as traditional paper maps do. This can lead to a more natural interaction [36].

Examples of mobile spatially-aware applications include: field work applications [197, 46]; tour guides, that help people navigate an unfamiliar space [43, 2, 138, 134]; navigation assistants [13]; time management applications [56, 64]; messaging applications [34, 216, 38]; office applications, such as nearest printer services [118, 204]; conference aids, that track presentation attendance and facilitate note taking and discussion [59]; and home applications that can help with household management and home entertainment, as well as aid the aged and disabled in performing everyday tasks [108, 114, 140].

In the current thesis we consider spatial information to have two components: 1) the spatial data, that is, the geometry and location of objects in the real world; and

2) the set of spatial operations used to analyse, manipulate and manage the data. Background on each of these topics follows.

Spatial Data

Spatial data describes the physical structure of the user's environment and relationships between real world objects [157]. Since ancient times, maps have been serving as a "favourite communication tool" to help people construct personal mental images of their living environment [142]. Indeed, maps are the most efficient and effective means to communicate spatial information [119]. They simplify the localisation of geographic objects, revealing spatial relations and patterns, and provide useful orientation information in the field [142, 37]. A common example of spatial data is a road map. A road map is a two dimensional visualisation of graphic information that contains points, lines, and polygons that can represent cities, roads, and political boundaries such as states or provinces. This information is used to judge distance and navigate[2].

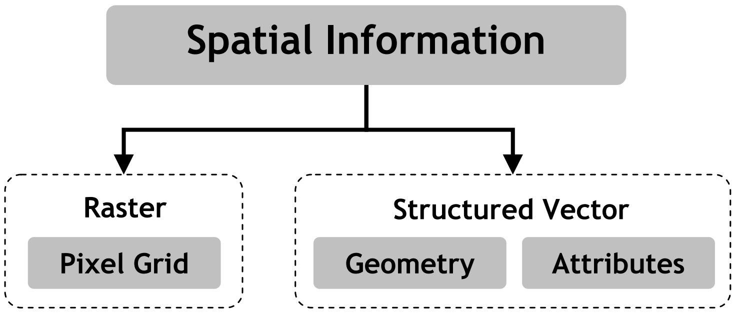


Figure 1.2: Types of spatial data model

Spatial data models can be grouped into a number of categories based on their underlying data structures. Figure 1.2 is a taxonomic classification of the different types of spatial data model. The primary classification is as *raster* or *vector*. Raster spatial data is essentially a paper map converted to electronic format. Information is structured as a grid of cells where each cell is analogous to a pixel in an image [89, 243]. In contrast, structured vector data is represented by a set of vector coordinates. The vector coordinates are organised into shapes such as point, line, and area features (that is, geometry). Attribute or database records may be associated with individual shapes [89]. Vector data facilitates geometric operations such as overlap, intersection,

[2]For a more detailed analysis of the affordances of maps see Meng & Reichenbacher [143].

containment and distance, which can be used as part of a spatial reasoning process to determine higher level contexts or trigger behaviour.

Spatial Operations

In addition to spatial data, mobile spatially-aware applications require algorithms to manipulate and manage the data in support of spatial services. Geographic information systems (GIS) have for many years provided this support on non-mobile devices. The characteristics and complexity of these spatial operations form an important background to the current thesis, as any manipulation of spatial data on a mobile device requires models and algorithms capable of providing lightweight versions of these services.

A GIS is defined as a system "capable of capturing, storing, analysing, and displaying graphically referenced information; that is, data identified according to location" [231]. More simply, a GIS can be thought of as a software system that links geographic information (where things are) with descriptive information (what things are like).

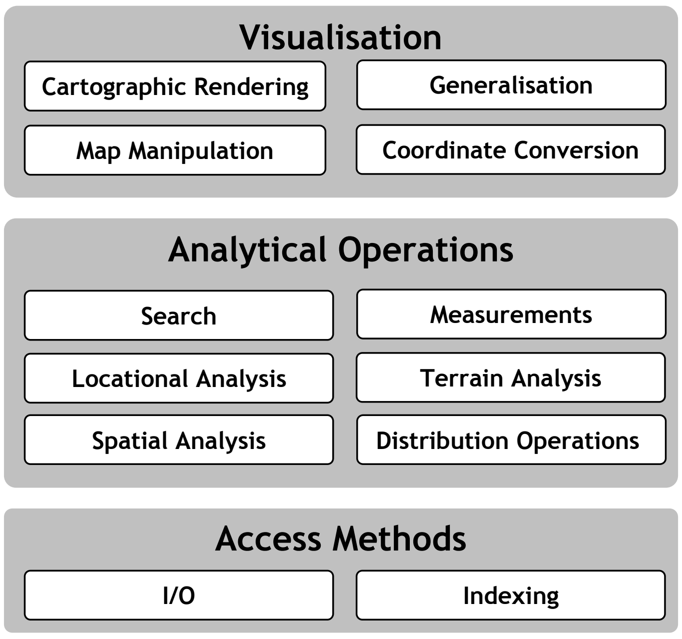


Figure 1.3: Categorisation of GIS spatial operations

Figure 1.3 presents a categorisation of the core GIS operations gathered from the literature [6, 210, 172, 190, 130]. *Access Methods* include all functions for interacting with spatial data formats. The collection of *Analytical Operations* includes: *Search* functions that deal with the retrieval of geometric data based on graphic or thematic constraints; *Location Analysis* refers to operations that overlay or extrapolate spatial information; *Terrain Analysis* covers operations dealing with 3D elevation data; *Distribution / Neighborhood Operations* determine connectivity, proximity and shortest path between features; *Spatial Analysis* describes the relationships and dependencies among spatial objects; and *Measurements* contains calculations (distance, direction, perimeter, area, height, etc.), statistics, and topological measures (adjacent, connected, inside, etc). The *Visualisation* class of GIS operations includes: *Generalisation*, the reduction of the number of points necessary to represent a feature; *Coordinate Conversion*, the transformation and projection of coordinates from various graphic coordinate reference systems; *Cartographic Rendering*, the drawing of map-based interfaces; and *Map Manipulation*, the interaction with and customisation of the interface with operations such as pan and zoom.

The development of an architectural model featuring algorithms capable of manipulating spatial data on a hand-held mobile device requires lightweight versions of GIS algorithms for each category of spatial operation to counteract the limitations described in Sections 1.1.1 and 1.1.2. This set of spatial operations are important as they provide the means to generate, manipulate and manage the spatial data required by spatial middleware services.

1.2 Motivation

The International Telecommunication Union has predicted that by the end of 2008, more than half the world's population will have access to a mobile phone [1]. Smartphones (mobile phones that run a complete operating system providing a platform for application developers) will be the most common programmable computers on earth in two or three years [170].

Widespread use of hand-held mobile devices, coupled with their increasing capabilities has substantially contributed to the increasing mobility of our working and everyday life. Mobile devices allow users to take care of business and social obligations throughout the day by being more 'connected' than ever before. They improve coordination and efficiency by eliminating wasted time between activities and while waiting for input from individuals who may be traveling [199]. Examples of the types of applications that are typical of today's mobile devices include communication (voice, email, SMS), personal information management (calendars, to-do lists, address books, bookmarks), navigation [88], gaming [83] and mobile versions of established web services such as Yahoo! oneSearch$^{\text{TM}}$[3] and Google Maps for Mobile[4].

Mobile devices are increasingly making use of sensing technologies to gather information on the environment in which they are being used. The situational information gathered from sensors and positioning technologies is referred to as *context* and has been defined by Dey as "any information that can be used to characterise the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and the applications themselves" [57]. Context-aware applications are applications that exploit knowledge of this situational information to guide their behavior, typically with the goal of improving the usability or efficiency of the application [55]. This is achieved by the application adapting its behaviour, to make computation useful in the myriad of situations that can be encountered in the real world [155].

A common feature of context-aware mobile applications is that they require some form of spatial information. For example: tour guide applications require knowledge about the connectedness of places on the tour and the ability to guide users with meaningful directions; field work applications need to be able to reference the location where samples were recorded; and time management applications need to be able to estimate the traveling time between activities.

Spatial information is central to much of the visualisation and reasoning about the user's environment in mobile computing. It is exploited by mobile applications to judge distance, determine visibility, produce routes and provide other useful services to mobile users. Spatial information can contribute to the usefulness of context-aware applications by providing a "deeper understanding of the physical space from both a sensory (input) and control (output) perspective" [35]. For example, fixing a position in some coordinate system to a certain degree of accuracy is not as useful as being able to determine if two entities are in view of each other [36]. Spatial information also has a role to play in supporting richer interactions with the user by acting as a shared

[3]http://mobile.yahoo.com/onesearch
[4]http://www.google.com/mobile/gmm

metaphor [36, 65] and in integrating context from different sources. For example, spatial information can be used to integrate sensor data in a way that enables analysis and inference [35, 43, 158, 185]. Numerous innovative applications such as location-aware resource management [202], real-world point & click [65, 85], landmark based navigation [139, 148], context-aware collaboration [164], de-cluttering map data [24], and information tailoring (intelligent proximity) [132] are only possible when spatial information is readily accessible by mobile applications.

However, the relative resource poverty of mobile devices as well as their lower trust and robustness points to reliance on static servers [200]. Additionally, the services required of mobile computing applications are traditionally provided by geographic information systems (GIS). GIS store spatial information, both geometry and attributes, provide access to this information, perform analytical functions and present the results visually as maps, tables or graphs. There are no fully featured mobile equivalents of these highly specialised GIS applications. The lack of a mobile GIS has confined spatial data to central servers, accessible to mobile devices in the form of a static graphic representation via wireless networks. This raster-based graphic representation cannot provide meaningful map entities, such as streets, buildings and dynamic objects, that can be interpreted by a mobile device. In addition, raster data has the further limitations of requiring more storage space and consequently transmission bandwidth than the same data in vector form. Raster data also makes the integration of map data sets from different sources practically impossible [222]. When access to spatial data is restricted to raster data, mobile applications require a round trip to the server each time dynamically changing context requires an interface update. In the case of mobile applications, such interface updates are frequent because of the small screen of a mobile devices forcing the user into a great deal of panning and zooming to visually link map features to objects in the environment [52].

In contrast, vector data is more compact and facilitates information browsing and analysis [100]. Vector-based spatial data facilitates metric operations such as overlap, intersection, containment and distance, which can be used as part of a spatial reasoning process to determine higher level contexts (for example, speed, indoors or outdoors, proximity to other users) or trigger behaviour (for example, proximity based information display, connect to a wireless network, toggle screen backlight). Vector-based spatial data enables the generation of new views and processing of some types of queries locally without having to request additional data from servers [33]. For example, zooming or panning actions are cumbersome, with performance depending on how much data the server must transfer every time the user requests a new view. Vector-based spatial data allows zooming or panning actions to be performed without communicating with a server.

Vector data also has the advantage of enabling the rendering of high quality cartographic interfaces that are scaled to suit the resolution and small size of mobile displays and clear enough to be readable under daylight conditions. Both these spatial data representations have been used by mobile applications, although raster spatial data is favored as it allows the computationally expensive analysis and rendering of cartographic interfaces to be done on the server [227, 230].

It is a significant challenge to implement lightweight versions of computationally intensive GIS operations on a mobile device because of the volume of spatial data to be managed, the computational complexity of GIS operations and the limitations of mobile devices [246]. Where current mobile spatially-aware applications feature vector-based spatial information, they are limited to visualising this information and do not support GIS operations [243, 161, 246].

The computational complexity and volume of spatial data also has an impact on energy consumption. The computation required to manipulate and analyse spatial information requires extended use of the device's processor, a major source of energy consumption on mobile devices. Alternatively, this computation could be performed on the server and the results delivered as a service to the application. However, this approach requires use of the wireless network which is another major source of energy consumption on a mobile device. One of the open questions in the area of low-power research concerns the proper balance between communication and computation, that is, whether to perform an operation locally or to have the operation processed remotely at the cost of communication [137].

To date, spatially-aware mobile applications have been developed by providing mobile applications with access to network spatial services via wireless networks [190]. When spatial services are based on the device's position, they are termed location-based services [177]. These services enable spatially-aware applications by providing, manipulating, analysing, communicating and visualising spatial information [144]. However, this approach can not provide constantly available spatial services to mobile applications due to frequent disconnections and unreliable nature of wireless networks.

Where spatially-aware mobile applications have provided spatial services such as routing, map-based interfaces and spatial reasoning, they have been tailored for specific devices or relied on infrastructure-based services [42]. The development of spatially-aware applications is further hampered by the lack of a generic model for spatial services that addresses the common challenges posed by hand-held mobile devices. These challenges (See Sections 1.1.1 and 1.1.2) are currently addressed by each individual application, resulting in the repeated näive reimplementation of these services [188]. A solution to this problem is to provide middleware containing implementations of basic GIS services for mobile spatially-aware applications, which is independent of application-specific services and particular sensors [200]. Providing these services in the form of middleware supports generality, making spatial functionality easy to reuse in other applications as there is a separation between application semantics and the low-level details of spatial data interpretation and cartographic interface production.

Indeed, there have been many efforts to develop middlewares, frameworks, toolkits, and infrastructures[5] to address the application independent challenge of managing context on mobile devices [103, 58, 64, 93, 129]. These infrastructures provide "uniform abstractions of common functionality and reliable services for common operations" to make it easier to develop robust applications, even on a diverse and constantly changing set of devices and sensors [105]. As the challenge of storage, analysis, management and visualisation of spatial information is also independent of application specific concerns, there has been some work in developing middleware to support these tasks [81, 246, 134, 222]. However, these middlewares for mobile applications assume reliable wireless connections and require significant infrastructure.

The main motivation for the current thesis is the need to provide constantly available spatial services on mobile devices to support the development and interoperability of spatially-aware applications using an approach that places energy conservation at the core of all design decisions. This thesis proposes an architectural model for a spatially-aware mobile application middleware. This model features algorithms desi-

[5]See Hong et al. for a discussion on the differences between these terms [105].

gned to minimise the processing time and power consumed on hand-held mobile devices while providing uninterrupted access to common spatially-aware application services, such as rendering of geospatial information, real time generalisation of dynamic scenes, route generation, and visibility determination.

In summary, the research questions are:

RQ-1 How can spatially-aware mobile applications maintain a dynamic model of the users' environment including meaningful map entities, such as, streets, buildings and dynamic objects in an interpretable format despite the inherent unreliability of wireless networks and limitations of hand-held mobile devices?

RQ-2 How can we design algorithms to access a spatial model at multiple levels of detail on hand-held mobile devices that are limited in terms of battery power, processing resources and memory?

RQ-3 How can common application-level spatial services be designed for mobile spatially-aware applications using the set of algorithms for spatial operations included in the model for spatial middleware?

RQ-4 What are the energy trade-offs in balancing computation with communication for this class of spatially-aware mobile application?

RQ-5 Is there a generic model for spatial services that can address the common challenges posed by hand-held mobile devices in a reusable and extensible manner?

1.3 A Model for Mobile Spatial Services

This thesis presents an architectural model for spatially-aware middleware, including the design of a set of algorithms that provide reusable lightweight variants of spatial services on hand-held mobile devices. The set of algorithms includes algorithms designed to minimise the processing time and power consumed on mobile devices, while providing uninterrupted access to common spatially-aware application services. The following sub-sections introduce the main algorithms in the areas of: 1) availability of spatial services (RQ-1); 2) reduction in complexity of spatial data (RQ-2); 3) support for mobile spatial services (RQ-3); 4) visibility determination (RQ-3); and 5) genericity

(RQ-5). Research question five (RQ-5) is addresses through a comparative evaluation, the design of which can be found in the evaluation chapter (Section 5.2.1 on page 156).

1.3.1 Constant Availability of Spatial Services

Spatially-aware mobile applications operate in an environment where mobility and the limitations of wireless networks restrict access to GIS-based network services. Despite the lack of a reliable network, mobile applications must be capable of performing access, analysis and visualisation operations on spatial data (Section 1.1.3). However, spatial data is dynamic, reflecting the changing user's environment, and spatially-aware applications are interactive in nature, placing further demands on the analysis and visualisation of spatial data. In order to meet the demands of a dynamic interactive environment, without compromising the user experience of spatially-aware applications, spatial middleware services must remain constantly available, despite the unreliable nature of wireless networks and resource limitations of hand-held mobile devices. The current thesis proposes an architectural model that takes the approach of distributing vector-based spatial data to hand-held mobile devices.

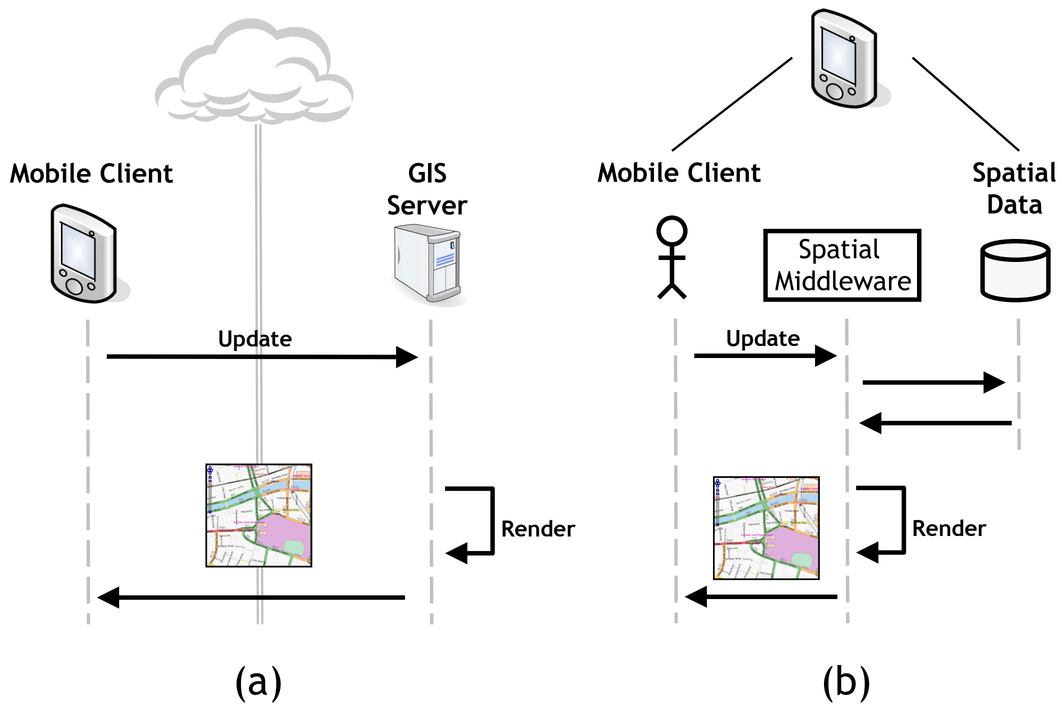


Figure 1.4: Comparison of architectural models

Figure 1.4 (a) summarises the state of the art in spatially-aware mobile applications where spatial services are provided by servers, with spatial data accessible in the form of raster maps. Figure 1.4 (b) illustrates the novel middleware approach proposed by this thesis. In this case the spatial data is stored locally on the hand-held mobile device. When a mobile application invokes a spatial service such as map rendering, the set of algorithms providing spatial middleware services on the mobile device perform the necessary spatial operations. This novel approach eliminates the dependency on network connectivity, enabling constant availability of spatial services to mobile spatially-aware applications. This increased availability of spatial information enables the development of applications featuring dynamic maps, as well as user authoring and annotation of maps, and spatial decision support [194]. This also has the advantage of reducing the energy consumed by the mobile devices, allowing for longer operation between charges.

1.3.2 Spatial Complexity Reduction

The architectural model for spatial middleware services takes the novel approach of locating spatial data on hand-held mobile devices and performing all spatial operations locally. However, spatial operations are computationally complex due to the volume of coordinate-based spatial data that they operate on. This challenge is addressed by a set of algorithms designed to reduce the complexity of spatial data, thereby reducing the processing time, memory and power consumed while manipulating it on mobile devices. These algorithms include a multi-scale, spatial data structure designed to approximate continuous scale adaptation with stepped levels of detail [98]. Access to this data structure is facilitated by a hierarchical spatial index, that uses minimum bounding boxes to approximate more complex spatial objects [188]. The levels of detail are derived using algorithms that eliminate objects based on rendered size at particular scales and simplify geometry using shape and location preserving 2D generalisation algorithms that remove vertices's by analysing directional trends against a tolerance factor [208, 181, 77]. These generalisation algorithms are model-oriented (that is, generalisation happens the spatial data model level, as opposed to at the graphic representation level [53, 131]).

The architectural model uniquely combines these algorithms with buffering strategies that minimise the impact of the rasterisation process by taking into account the static nature of the majority of spatial information. Map projection algorithms, used to convert from longitude and latitude coordinates to a mobile devices screen coordinate system, are chosen to optimise the performance of coordinate transformation at the cost of acceptable area and shape distortion [48]. Limited processing resources are preserved by clipping geometry that extends beyond the viewport of the device, to avoid computing projections and rendering coordinates that will not be seen. The novel architectural model proposed in this thesis combines and integrates these algorithms into a spatial middleware service capable of selecting, retrieving and rendering spatial data on hand-held mobile devices.

1.3.3 Spatial Services

Mobile, spatially-aware applications require a number of spatial services. These services include: 1) Routing; 2) Cartographic Rendering; and 3) Spatial Reasoning. The architectural model for spatial middleware services proposed by this thesis contains algorithms that provide support for each of these services.

Routing assists spatially-aware applications in generating routes between locations. To achieve this, applications require an understanding of the topological[6] structure of the user's environment. Topological representations of regions are a graph-based alternative to metric maps (also called Euclidean maps) where nodes are features and edges are paths between them. Spatial information uses an absolute coordinate system and numerical measures of positions and dimensions of objects to describe features of the environment. Representing and generating routes based on metric maps is memory and processor intensive. However, they are fast to build, easy to update, and translate well to graphical representations making them suitable for use on mobile devices. This thesis proposes an architectural model where topological relationships between objects are determined from spatial data describing the user's environment as needed.

Cartographic rendering supports mobile applications in communicating spatial information to the user through their interface. In the case of location-aware applications, locations must be communicated to the user along with any context that may impact the user. A cartographic (map-based) interface is a method of communicating context to mobile users that is commonly found in mobile location-aware applications

[6]The topological structure refers to the logical structure of areas based on spatial relationships (connectedness).

[129, 93, 50]. Location-aware applications lend themselves to a cartographic style of explanation, in which relationships between locations are depicted diagrammatically in terms of relative proximity. The architectural model proposed by this thesis contains an algorithm that produces cartographic interfaces at varying scales and resolutions, on top of which contextual information can be rendered (drawn to the screen).

Spatial reasoning is the determination of metric, topological and logical relationships between locations. Mobile applications rely on spatial decision support to locate nearby services, deliver directions or estimate journey times. Spatial data can be used by spatially-aware applications as a common frame of reference to facilitate better collaboration between peer devices. As an example, imagine mobile users who are equipped with a camera that is used to determine location ("I can see that I am in front of the gate"). With the ability to access and reason over spatial data it may be possible in the future for users who are in sight of each other to collaborate to help determine each other's positions ("I can see that the door is behind you"). From this example it is seen that being able to fix a position in some coordinate system is not as useful as being able to determine if two entities are in view of each other [23]. An understanding of physical relationships between physical entities, for example walls block users' visibility, will be necessary in future location-aware applications [36].

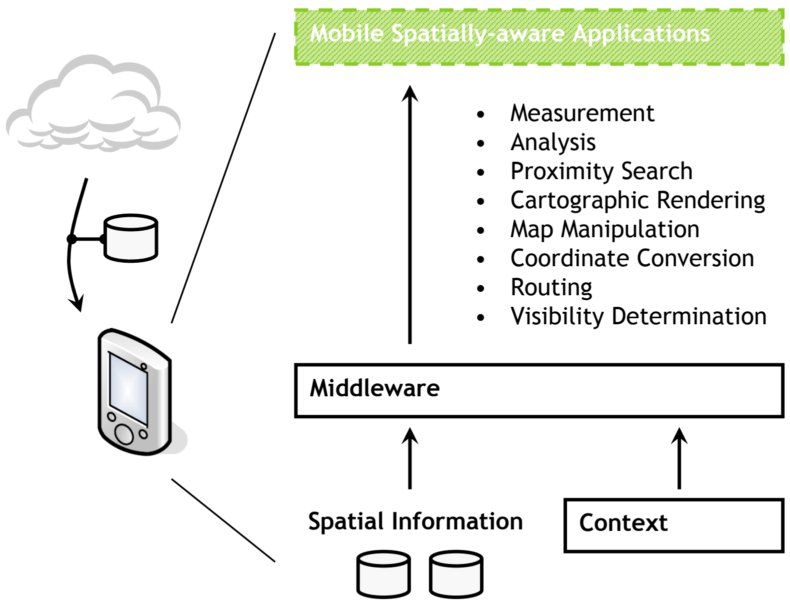


Figure 1.5: Middleware model for spatial services

Figure 1.5 summarises the spatial middleware services provided to spatially-aware mobile applications. In order to support these services the architectural model provides algorithms for each of the GIS operations for manipulating spatial data presented in Section 1.1.3.

1.3.4 Visibility

The ability to determine if two entities are in view of each other is an example of an innovative spatial service that is supported by the architectural model described in this thesis. This service requires knowledge about features in the real world to compute whether an object is something that the user can see. This knowledge can be used in navigation applications to present instructions based on features that are visible in the real world. This is only possible with access to the vector-based spatial information that is constantly available and of reduced complexity [23].

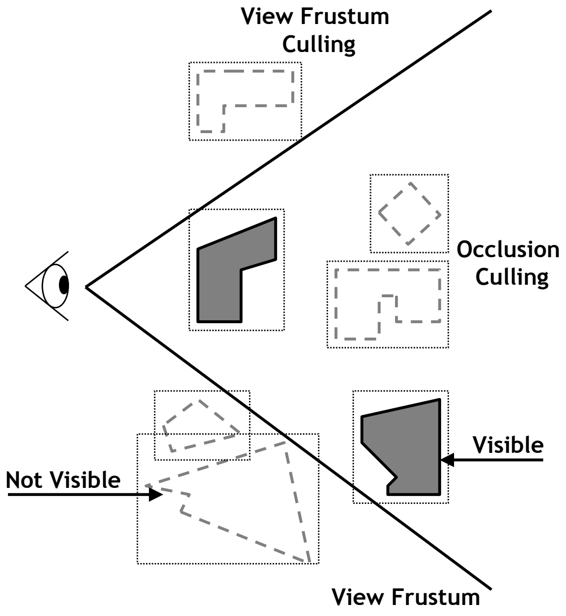


Figure 1.6: Illustration of visibility culling techniques

Object precision visibility is determined by filtering objects within the application's viewport based on field of view and occlusion. The objects are stored in a depth buffer [78] (linear data structure ordered by distance from point of view) on which visibility culling techniques including view frustum culling and occlusion culling are applied (Figure 1.6) [240, 244]. In order to reduce the demand on computational resources and energy, the algorithms use bounding boxes of individual objects to approximate their shape. This algorithm reduces the computational requirements of determining visibility of spatial objects by finding a conservative estimate as opposed to an exact answer to the problem.

1.3.5 Genericity

The model for spatial middleware, including the set of algorithms and spatial services, provides a design whose implementation supports developers by providing generic structure and behaviour that addresses the common challenges in mobile spatially-aware applications. The model is generic, as it provides spatial services that are not confined to a single application domain but may be reused by developers of a range of mobile applications. For example: field work applications, tour guides, navigation assistants, time management applications, messaging applications and conference aids (Section 1.1.3). Resuse of the model is also directly supported by a reusable implementation that provides reusable and extensible mechanisms for spatial data management and spatial operations that execute on a mobile device. Extensibility in the model is supported by the pluggable component model that defines specific extension points where alternative algorithms may be substituted. The substitution of component implementations is facilitated using a configuration file that specifies the component in use and its parameters.

1.4 Contribution

The main contributions of the thesis can be summarised as follows:

C-1 (Addressing RQ-1)

A model for mobile spatial middleware that combines a vector-based spatial model with the ad-hoc collaboration features of the Hermes framework for mobile computing to autonomously disseminate spatial data to mobile devices.

C-2 (Addressing RQ-2)

A multiple representation database that dynamically generates levels of detail via model generalisation. Continuous scale adaptation is approximated with stepped levels of detail and retrieval and insertions are facilitated by a hierarchical spatial index that uses minimum bounding boxes to approximate more complex spatial objects.

C-3 (Addressing RQ-3)

A set of algorithms for manipulating spatial data to support services such as: route generation; rendering of personalised, adaptable map-based interfaces; and spatial reasoning. This set of algorithms preserve limited processing resources, satisfying users' expectations, by selecting data at an appropriate scale, clipping geometry to viewport extents, buffering static objects and caching coordinate transformations.

C-4 (Addressing RQ-3)

The design of an innovative visibility determination service for spatially-aware applications based on the use of a depth buffer on which a variation of occlusion culling is performed to reduce the set of geometry to a possibly visible set (PVS).

C-5 (Addressing RQ-4)

A comparison between the model for mobile spatial services presented in this thesis and existing approaches investigates the energy trade-off when rendering and interacting with map-based interfaces to spatially-aware mobile applications.

C-6 (Addressing RQ-5)

A generic framework for spatial services designed not to overburden the limited resources of hand-held mobile devices and does not depend on continuous network connectivity.

Two approaches have been taken to evaluating the thesis contribution. Performance evaluations demonstrate that spatial services (map rendering, generalisation, route generation and visibility determination) can be provided locally on hand-held mobile devices. In addition, empirical experiments compare the trade off in power consumption between performing these spatial operations locally versus off-loading the tasks to a server. The contribution as regards the reusability of the model for spatial middleware services to support the development of a range of mobile, location-aware applications is evaluated through the development of a case study application.

1.5 Thesis Outline

The reminder of this thesis is organised as follows. Chapter 2 presents an overview of the state of the art in the areas of spatial services for mobile computing, commercial GIS and web mapping. Chapter 3 describes the design of a model for spatial middleware, a local environment model, and spatial middleware services built using the model. Chapter 4 presents the implementation of the environment model and spatial services, describing how these components are integrated to form a generic framework for spatially-aware applications. In Chapter 5 the performance of the spatial services are evaluated, the results of experiments investigating the energy trade-offs in performing spatial operations locally versus off-loading the tasks to a server are presented, and a case study spatially-aware application demonstrates the reusability of the framework. Chapter 6 concludes with a summary of the most significant contributions of this thesis, commenting on the overall benefit of the approach and discusses research issues that remain open for future work.

Chapter 2

Related Work

This chapter assesses research related to mobile spatial services. The extent to which each of the related projects addresses the research questions, identified in Section 1.2, is discussed. A review of related work in the following areas is provided (Figure 2.1):

1. Spatial services for mobile computing.
2. Map-based mobile services.
3. Commercial GIS & web mapping.

The systems presented in this chapter are representative of a wider range of systems, chosen based on their unique features or their influence in the development of the field.

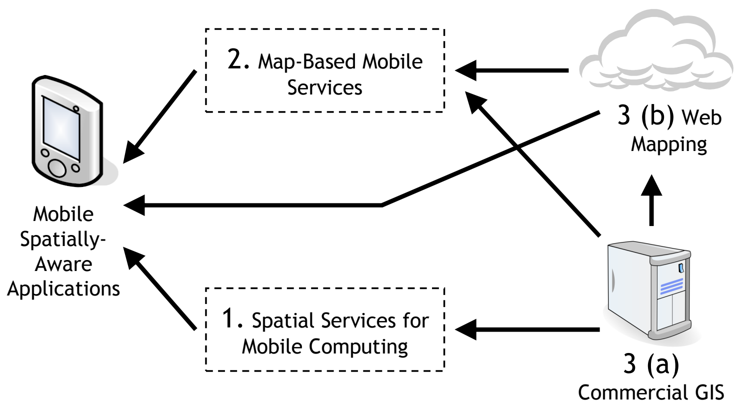

2. Map-Based Mobile Services
3 (b) Web Mapping
Mobile Spatially-Aware Applications
1. Spatial Services for Mobile Computing
3 (a) Commercial GIS

Figure 2.1: Technologies related to this thesis.

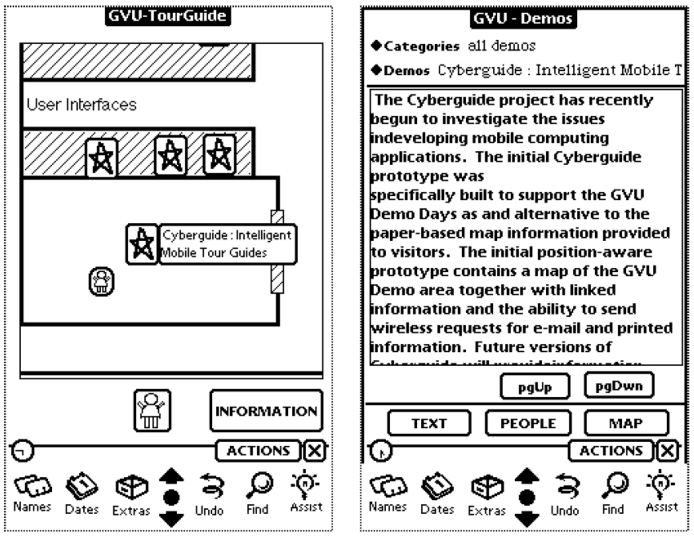
2.1 Spatial Services for Mobile Computing

The projects included in this section on spatial services for mobile computing are those that provide application frameworks, middleware or infrastructural support to the development of spatially-aware mobile applications.

Mobile spatially-aware applications incorporate features or services that require knowledge about the logical and topological structure of the user's environment. These applications run on hand-held mobile devices and typically make use of a positioning system to identify their location in the real world [157, 31]. Several research projects have produced mobile spatially-aware applications: comMotion [136], GeoNotes [68], Riot! 1831 [192], for example, are based on the common idea of attaching digital information to real-world places like a virtual post-it note or graffiti. Other projects have focused on mobile tourist guides as an application domain (for example, Guide [43], REAL [19], TellMaris [120]). Mobile games (for example, Can You See Me Now? [26]) have also enabled the exploration of the mobile use of spatial information. These applications are classed as spatially-aware as they provide some spatial services to their users. The range of services includes: navigation [43, 19, 120], map-based interfaces [43, 19, 68, 26] and location specific information services [120, 26, 192, 136]. These application services are supported by computation incorporating spatial operations such as containment, proximity, coordinate conversion, map rendering, spatial selection and route generation.

This section reviews systems that provide one or more spatial services to spatially-aware mobile applications including map-based interfaces, spatial information delivery (Deep Map [134], GiMoDig [222], M-Spaces [200], GinisMobile [183]), navigation (CRUMPET [178], Nexus [165], Lol@ [179]), adapting spatial information for use on mobile devices (GiMoDig [222]) and visibility determination (Point to Discover [84]). Location specific information services are excluded from this list as they require a network connection, making it impossible to provide uninterrupted access to this service in an unreliable wireless networking environment. The services for adapting spatial information for use on mobile devices and visibility determination are reviewed, because these services are included in the model for spatial middleware proposed by this thesis.

Systems providing spatial services for mobile computing are compared below ac-



The Cyberguide project has recently begun to investigate the issues indeveloping mobile computing applications. The initial Cyberguide prototype was specifically built to support the GVU Demo Days as and alternative to the paper-based map information provided to visitors. The initial position-aware prototype contains a map of the GVU Demo area together with linked information and the ability to send wireless requests for e-mail and printed information. Future versions of

Figure 2.2: The Cyberguide MessagePad interface (Source: Abowd et al. [2])

cording to several criteria based on the research questions defined in Section 1.2:

1. *Availability;* The availability of spatial services, despite the unreliability of wireless networks (RQ-1).

2. *Limitations of Mobile Devices;* Design features, algorithms or architectural components that specifically deal with managing or adapting to the limitations of mobile devices (RQ-2).

3. *Spatial Operations;* The set of spatial operations and services provided and the significance of their location within the architecture (RQ-3) .

4. *Genericity;* The generality, reusability and extensibility of the system (RQ-4).

2.1.1 Cyberguide

Cyberguide is a project of the Future Computing Environments Group at Georgia Institute of Technology, Atlanta, USA [129, 2]. Cyberguide's goal was to build prototypes of mobile tour guides using commercially available hardware (Figure 2.2). These prototypes enabled the investigation of issues common to location-aware application development in mobile environments. Cyberguide (the most cited of a range of similar location-aware tour guides) addresses how spatial data is stored, queried and represented to the user on a mobile device.

The Cyberguide system has a number of components, namely: *Cartographer*, a map component that handles map representation on the mobile devices; *Librarian*, responsible for delivering information on places of interest to the tourist; *Navigator*, responsible for locating the tourist's physical position indoors or outdoors; and *Messenger*, allows the user's position to be transmitted to a central service that could then help tourists find each other or facilitate the broadcasting of messages to all users at a certain place.

The above components were designed to run on hand-held mobile devices, although the communications component required network access to a messaging server. The information component stores all information on the mobile devices (including maps), making it difficult to propagate changes in information as each device must be individually updated.

Availability

As Cyberguide's *librarian* component stores all information on the device, a wireless network connection is not required. Storing all the information on the device allows constant availability of applications at the cost of making information difficult to update. Although Cyberguide stores all information on the device, a requirement for future mobile applications to access storage resources through "substantial communication and networking resources" was identified [129]. To investigate this possibility, a serial IR network, similar to that used by REAL [19], was built. This inexpensive network was used to communicate events and support messaging but not to access application specific information or spatial information. The inability to access spatial information means that changes (for example, the change of label in a map) still need to be manually copied to each device.

Limitations of Mobile Devices

Cyberguide is designed to work with the Apple MessagePad 100 and pen-based PCs running Windows for Pen Computing 1.0. Prototype applications for the Cyberguide system were designed as mobile applications from the beginning, with their functionality limited only to what was achievable in this hardware and software environment. For example, services such as an adaptable map-based interfaces or route generation were not possible because of their computational complexity and the limited computation resources of the Cyberguide hardware.

Spatial Operations

Cyberguide has a raster map-based interface that automatically scrolls based on the user's position (Figure 2.2). Information is overlaid on this interface in the form of map markers. Despite the project name, Cyberguide involves little guidance, navigation support, or route generation. This is largely due to Cyberguide not maintaining a model of the real world, limiting its ability to generate paths or highlight objects. Cyberguide is also limited to a small geographic area and requires recompilation to operate in a new location.

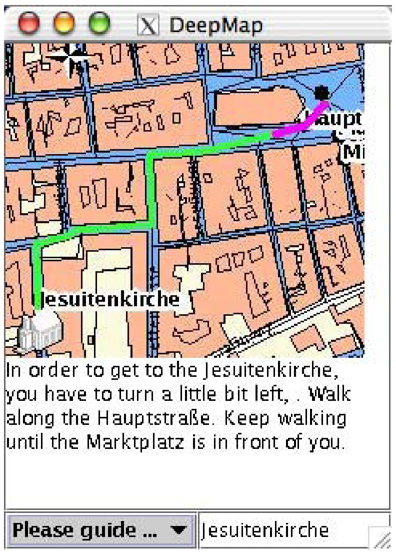
The Cyberguide project also experimented with a vector-based map representation. This feature was found to support manipulation and additional services such as way-finding, but was not used because the computation required to generate a display was greater than that available in the hardware deployed (Apple MessagePad 100). Vector data was also difficult to obtain and the size of the vector-based map database prohibited local storage on hand-held devices [2].

In contrast, the bitmap representation was found to be easy to obtain (scanning) and relatively inexpensive to store and display. However, scaling and rotation are computationally expensive using a bitmap representation. In addition, the bitmap representation lacked accuracy with respect to the real-world and was not suited to providing higher level map services, such as, generating a path to direct the tourist to a location of interest [2].

Genericity

Cyberguide is designed as a system whose conceptual design rather than implementation would be reused by developers. The utility of Cyberguide's architectural decomposition stems from the extensible and modular approach taken to system development. It is extensible as further services can be added and modular because it allows one component to be changed without impacting the rest of the system.

Cyberguide introduced the concept of allowing a user to navigate through a collection of activities using a mobile device, location tracking and map-based user interface.



In order to get to the Jesuitenkirche, you have to turn a little bit left, . Walk along the Hauptstraße. Keep walking until the Marktplatz is in front of you.

Figure 2.3: Deep Map interface (Source: Kray [120])

Subsequent mobile, spatially-aware applications (notably GUIDE [43]), including those that are developed using the model and algorithms for spatial middleware services described in this thesis, are built on the concepts introduced in the Cyberguide system.

2.1.2 Deep Map

In the Deep Map project, an interdisciplinary research group at the European Media Lab developed a prototype of a digital personal mobile tourist guide for the city of Heidelberg, Germany [134, 120]. The goal of the project was to develop information technologies that can handle heterogeneous data collections and complex functionality but are still accessible to untrained users. Unlike Cyberguide, where all information is stored locally on the device, Deep Map takes a two tier, client/server approach with a commercial GIS acting as the server platform. An ArcView GIS server platform hosts a number of Java GIS-agents. Mobile clients communicate with these agents using an XML-based messaging system. Two clients exist for the Deep Map system, one is a web-based planning and exploring tool for virtual visits or pre-trip planning and the other is a mobile guide designed for a wearable computer (Figure 2.3).

Availability

The platform presented by Deep Map relies on a reliable network connection between infrastructural components and the mobile client. However, the Deep Map project raises a number of interesting research questions. In particular, Zipf ("Adaptive context-aware mobility support for tourists", an essay included in [218]) recognises the need to reduce dependency on network availability and concludes that "we must develop lightweight versions of these [GIS] components" in order to maintain a dynamic model of the users environment on mobile devices.

Limitations of Mobile Devices

The mobile Deep Map prototype is developed for a wearable computer (Xybernaut MA IV) with a hand-held color LCD display and a headset[1]. Deep Map deals with the processing limitations of mobile devices by off-loading complex spatial operations to a server. The server responds with raster map data. This approach trades the cost of energy usage and application responsiveness on the mobile device with communication costs, and assumes a reliable, always available wireless network connection. The performance or energy usage of this approach was not quantified.

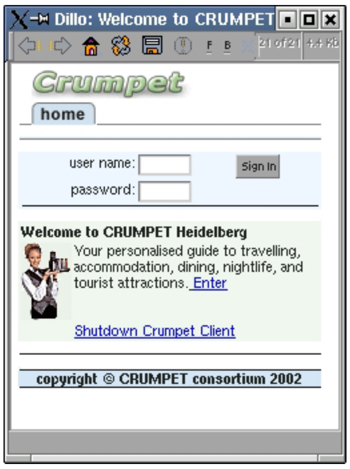
Spatial Operations

Spatial operations in Deep Map are handled by GIS-agents implemented using Java wrappers that communicate with the ArcView GIS via remote procedure call. One of these agents, the geo-spatial agent, retrieves spatial information from the GIS and performs a range of spatial operations [134]. Other agents are responsible for map rendering navigation and route finding. However, these spatial operations are all performed on server-side components of the Deep Map framework requiring continuous network connectivity with mobile clients.

Genericity

The Deep Map applications are limited to an area around Heidelberg castle where tourist information, mapping and wireless network coverage is available. Although the applications cannot be directly reused, Deep Map itself is considered a framework. The goals of the framework are to demonstrate that standard GIS functionality can implement personalisation and context awareness through adaptive map generation

[1]The Xybernaut MA IV is a lightweight computer with all the functionality and connectivity of a desktop computer. The CPU was a 200 or 233 MHz Intel Pentium MMX. The device had between 32 and 128 MB RAM and contained several GBs hard disk storage.


Dillo: Welcome to CRUMPET

Crumpet

home

user name:

password:

Sign In

Welcome to CRUMPET Heidelberg

Your personalised guide to travelling, accommodation, dining, nightlife, and tourist attractions. Enter

Shutdown Crumpet Client

copyright © CRUMPET consortium 2002

Figure 2.4: The CRUMPET welcome page (Source: [206] Figure 4.)

and personalised tour proposals. Although Deep Map demonstrated that GIS functionality has a place in personal mobile tourist guides, the research framework they propose is not generic or reusable, in that it is only designed as a testbed for prototyping applications in a controlled environment. Deep Map does not support the building of spatially-aware applications for deployment in the real world [134]. Several follow-on projects continued the investigation into mobile information delivery systems for tourists. One of these is CRUMPET.

2.1.3 CRUMPET

CRUMPET is an EU funded research project that followed on from Deep Map [178]. The goals of the CRUMPET project were to provide new information delivery services for a far more heterogeneous, mobile, tourist population. The CRUMPET project took a multi agent infrastructure approach to providing spatial services to mobile applications. Mobile devices host *terminal agents* that are responsible for providing the user interface (Figure 2.4). Network agents manage the communications layer and service agents wrap existing e-tourism services. Like Deep Map, spatial data is provided by a spatial agent that wraps an OpenGIS-Server. A service broker agent allows terminal agents publish interest in particular services and receive information about services that meet criteria such as proximity constraints. Using this model, services are combined on demand and adapted by agents to the user's environment to provide mobile applications with spatial information.

Availability

CRUMPET is an infrastructure platform and requires reliable wireless connection to deliver spatial services to mobile devices. The periodic disconnections that arise are accounted for through the provision of a location-aware service broker agent. This agent can deliver appropriate services to the mobile device based on its location. In the case of a disconnection due to mobility, this agent may be used by the mobile device to discover services at its new location. Mobile devices do not maintain a model of the users' environment and rely entirely on a reliable wireless network infrastructure to access the CRUMPET infrastructure.

Limitations of Mobile Devices

CRUMPET targets lightweight terminals such as next-generation mobile phones and PDAs, but to date, all field trials have been conducted using iPAQ hand-held computers. One of the goals of CRUMPET is to provide a service delivery platform that can adapt to the limitations of various wireless networks such as High Speed Circuit Switched Data (HSCSD), General Packet Radio Service (GPRS) or Universal Mobile Telecommunications System (UMTS). The CRUMPET platform includes functionality to monitor the quality of service of data transmission, control data transmission formats and select appropriate protocols. CRUMPET takes the computational limitations of mobile devices into account through its choice of a client/server approach, which requires a reliable network connection to maintain the user experience. Network communication consumes energy, a limited resource on mobile device. However, CRUMPET does not factor energy consumption on the mobile device into decisions regarding network interface, data transmission formats or protocols.

Spatial Operations

CRUMPET adopted OpenGIS Consortium standards for providing access to heterogeneous geographical data and geo-processing resources in a networked environment. These standards are the Web Mapping Interface Specification and the Geographic Markup Language (GML). GML is an XML version of the OpenGIS-Simple feature specification, which is a specification for vector based map-content for GIS systems.

The design approach is to make use of distributed geodata servers and corresponding web map servers for spatial information within the CRUMPET platform. However, applications developed for CRUMPET could not be considered spatially-aware as they do not consume vector-based spatial data but rely on the spatial agent to perform spatial functions such as spatial queries and selections, distance measures, exports of geometry data and visibility analysis. This means the CRUMPET applications do not include the spatial operations and algorithms necessary to maintain a model of the user's environment and can not provide access to common spatial services without the assistance of CRUMPET's web map servers.

The CRUMPET project recognised geographical information as being a crucial feature of tourism applications. CRUMPET concluded from its user trials that "Maps rank very high, and even higher when they indicate the current position of the user. The rendering of maps, enhanced by context-sensitive information, is probably a core feature for mobile tourism support" [206]. The same field trials also uncovered a requirement that maps on mobile devices should adapt to the user's orientation and include information such as building outlines, accessibility and bus stops. Although these findings were to influence the design of other mobile spatially-aware applications including the spatial services presented in this thesis, CRUMPET's implementation is server-centric and does not take into account the combined challenges of mobility, resource limitations and unreliable wireless networks.

Genericity

CRUMPET supports the development of spatially-aware mobile applications by basing its architecture on a standards-compliant open source agent framework. In addition, a standards-based approach to spatial services is taken and an implementation of the platform is available as open source code.

Strategies proposed by the CRUMPET project to reduce network dependence include intelligent pre-fetching and caching in combination with location awareness and resource (network, CPU) adaptivity. First steps to evaluating these proposals resulted in a prototype providing spatial operations such as topological queries on a PDA using R-Tree spatial data indexing [207]. However, further details of this prototype have not been published.

One of the outcomes of the CRUMPET project that has been highly influential on this thesis is the argument by Zipf [246] that spatial services need to run on mobile devices to reduce dependency on network availability. The model for spatial middleware presented in this thesis locates all spatial information locally on the mobile device. This design decision requires that the algorithms for manipulating the spatial information, the spatial operations for inferring relationships between objects in the environment and the spatial services such as route generation and map rendering, must also run on mobile devices. These challenges form the core areas of investigation in this thesis.

2.1.4 Nexus

Nexus is an infrastructure-based GIS platform for mobile spatial services developed by the University of Stuttgart [161, 163, 81, 165, 82]. The main focus is on modeling heterogeneous spatial information for a wide range of applications and offering an open platform on which other service providers can build their location services.

The Nexus platform specifies basic services, open protocols and open data exchange formats needed for location-based services. The central feature of the platform is its model of the real world called the Augmented World Model (AWM). Nexus is a three tiered architecture (Figure 2.5). Applications run on a mobile (but not necessarily hand-held) client and communicate wirelessly with the Nexus infrastructure [164].

Availability

Mobile, spatially-aware applications built for the Nexus platform (for example, NexusScout [162], CityNav [82], Virtual Information Towers [229]) rely on reliable wireless network connectivity between mobile devices and the infrastructure. Should this connection be interrupted, the application will no longer have access to any spatial data, spatial services or location-based events.

Limitations of Mobile Devices

Nexus makes no attempt to provide spatial data tailored for mobile devices. The responsibility of controlling the restriction of spatial information is left to the developers of Nexus applications. Nexus clients are Java Applets, developed for a notebook computer based on a web browser [125]. The Nexus platform is not suitable for

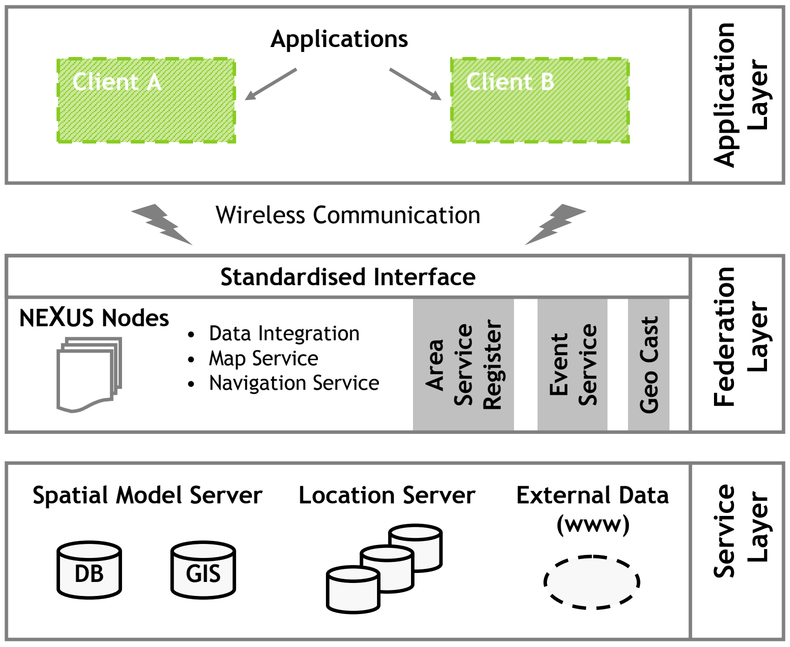


Figure 2.5: The Nexus architecture (Redrawn from [82], Figure 1)

spatially-aware applications on hand-held mobile devices due to the overhead of parsing the AWM modeling language into an object-oriented data structure. Like Deep Map, Nexus provides server-side implementations of common spatial services (for example, map creation, navigation) allowing these tasks to be offloaded from resource limited mobile devices.

Spatial Operations

The Nexus application layer provides tools for constructing queries for spatial data in AWQL (Augmented World Query Language) and tools for parsing the resulting spatial data back into instances of AWM objects. However, there is no client-side support for inference or deriving spatial services from this object-oriented data structure. The Federation Layer (Figure 2.5) provides support for spatial operations, including position, range, and nearest neighbour queries. In addition, a data integration service (from multiple Spatial Model Servers), a navigation service and a map service are provided.

Genericity

Nexus supports the development of spatially-aware applications by providing infrastructure that can deliver spatial information to mobile devices. Spatial information is accessed via a query language (AWQL) that supports object retrieval within the AWM

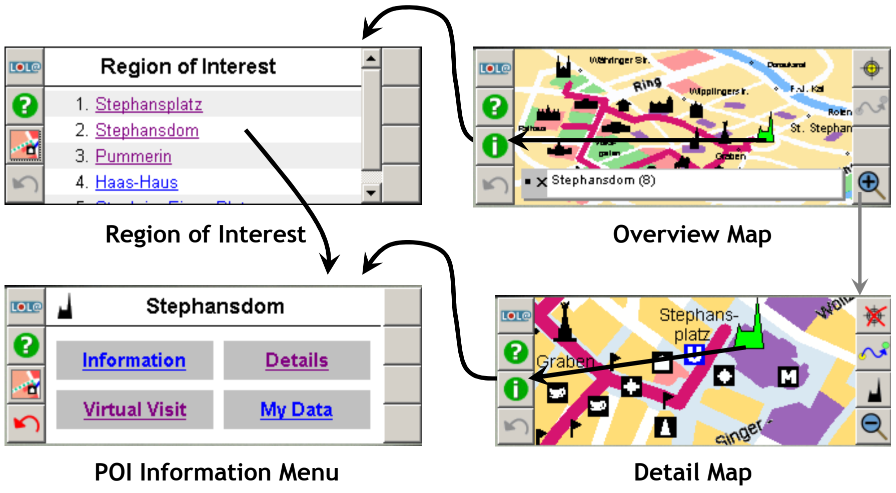


Figure 2.6: Lol@ map interface (Source: Umlauft et al. [223])

and a data exchange format called AWML (Augmented World Modeling Language) to serialise the objects of the AWM. In addition the Nexus project has published papers detailing a step-wise process of building spatially-aware applications using the Nexus Augmented World Model [165]. However, to date there has been no deployment of Nexus infrastructure.

2.1.5 Lol@

The Lol@[2] system is a mobile tourist guide for the city of Vienna, developed at the Forschungszentrum Telekommunikation Wien [89, 179, 223]. The tourist guide is a location-based mobile application designed for mobile phones that offers maps, localisation, routing functionality, and speech input.

LoL@ takes a similar architectural approach to CRUMPET. The main interaction metaphor is that of a browser, where a user clicks links or buttons to access information. LoL@ uses an Internet application architecture based on the client/server paradigm. A ‘terminal’ with permanent network connection accesses a remote server where most processing takes place. The client is responsible for the display and direct interaction handling on the mobile device through a web browser (Figure 2.6).

[2]Local location assistant

Availability

Although LoL@ takes a predominantly client/server architectural approach, it is designed taking technical constraints such as limited bandwidth and possible loss of network connection into consideration. LoL@ stores all content in a database on the server. However, unlike most client/server architectures, LoL@ moves some of the application's logic to the mobile client. Simple interactions like a button press, which do not cause an information update from the server, can therefore be handled directly within the device. This concept improves response time and reduces network load. Although LoL@ is capable of providing a better user experience than other client/server systems in a unreliable wireless network, LoL@ does not move enough functionality to the mobile devices to allow it support highly interactive mobile applications. Due to the dynamic nature of the user's environment, spatially-aware applications require reliable access to common spatial services and LoL@ does not achieve this goal as spatial information and services remain on the server-side.

Limitations of Mobile Devices

LoL@ is designed to account for resource limitations in a static way [179]. To accommodate the small screen size of mobile devices, the presentation of information uses the browser metaphor and the maps are carefully designed to remain usable at low resolutions and small sizes. Uncertainty in user position is accounted for by identifying location in the interface as a circular region rather than a point, where the radius of the region is an indication of the estimated error. Limited bandwidth is managed through a push-based content delivery and by controlling the frequency of refreshing the displayed maps based on the user's average moving speed and the chosen scale on the display. LoL@ does not feature algorithms to minimise the impact of other limited resources such battery power, processing resources or memory. Although LoL@ divides its logic between client and server, the division of responsibility does not take into account the energy trade-off in performing computation locally versus communicating tasks to servers. This limits the usefulness of the application as mobile devices will not operate for long between charges.

Spatial Operations

LoL@ supports a limited set of spatial operations. A map viewer applet in the terminal is able to display maps of the inner city of Vienna in two zoom levels (overview and detail). All detailed maps are designed for an on-the-fly production, including user-dependent display of information layers (route, point of interests) and a simplification of map elements. Coordinate transformations are done by a location server, while map preparation and route calculations are performed by a routing server. Although all the spatial operations in LoL@ are provided by servers, LoL@ successfully demonstrates the feasibility of using a map representation on smart phones with small displays. However, LoL@ does not provide mobile devices with access to a local spatial model, the model that is available on the server-side is limited to two levels of detail which is insufficient for most spatially-aware applications as it places restrictions on the ability to scale the interface; a key requirement on devices that are characterised by small displays.

Genericity

The server side platform of the LoL@ system contains several modules that could be reused by other applications. However, developer support for reuse of this technology was not a research goal of LoL@. The LoL@ systems targets the tourist guide application domain and does not provide generic spatial services for spatially-aware mobile applications.

2.1.6 GiMoDig

GiMoDig[3] is a project from several European national mapping agencies, funded from the European Union [208, 222]. GiMoDig investigated methods for delivering spatial data to mobile users from national primary geo-databases by means of real-time data-integration and generalisation. Although the main concern of this project was data integration and the delivery of spatial data to mobile devices, work was also done on production of personalised maps in a real-time environment and methods for progressively transmitting vector data and information presentation on small-display

[3]Geospatial info-mobility service by real-time data-integration and generalisation `http://gimodig.fgi.fi/`

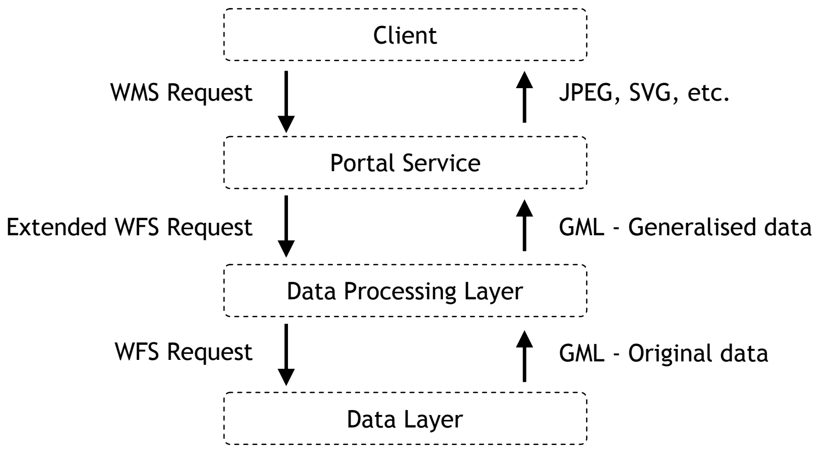


Figure 2.7: Simplified GiMoDig architecture (Source: Sarjakoski et al. [222])

mobile devices. The GiMoDig project takes a service-oriented approach to distributing cartographic data from core databases at national mapping agencies to mobile devices. Figure 2.7 illustrates a simplified overview of the GiMoDig system architecture. The architecture makes extensive use of Open Geospatial Consortium (OGC) Web-standards.

Availability

The GiMoDig platform requires a reliable network connection to operate. Although the spatial data delivered to the mobile device can be in SVG vector graphic format, it is designed for display and is stripped of all original GIS feature attributes. Because of this, the data can not support spatial operations (route generation, selection, restriction, information overlay, etc.) on the mobile device. However, simple vector graphic manipulation such as rotating and zooming is possible although any transformation of geometry can not be mapped back to the real-world coordinates of corresponding objects.

Limitations of Mobile Devices

The GiMoDig platform is designed to support mobile phones or PDAs. It is assumed that these devices are constantly connected to a network and have access to a positioning system and navigation service [168]. The limitations of these mobile devices are accounted for by: 1) placing all the spatial operations on the server-side and 2) performing real-time generalisation of spatial data in order to reduce the complexity of the map-based interface. The reduction in complexity of spatial data results in more efficient data transmission and a map that can be adapted and personalised to a specific device and individual user.

Spatial Operations

The GiMoDig project's goal was to generalise complex spatial data in order to provide a flexible presentation mode for spatial information on small-display devices. In GiMoDig, the spatial data is confined to the server-side platform with mobile clients having access to SVG or JPEG map images for display. GiMoDig considered the generalisation process "too complex to be executed in real-time" and "incapable of meeting the response time requirements for flexible zooming and adaptation" on mobile devices [222]. Hence, the GiMoDig service performs generalisation on behalf of the mobile device. In fact, all spatial operations in GiMoDig are performed within the server-side infrastructure.

The spatial operations supported by GiMoDig include map rendering, coordinate transformation, spatial data selection and generalisation. Real-time generalisation is the most significant contribution of the GiMoDig project. The real-time generalisation functionality implemented in GiMoDig includes the following generalisation operators: feature selection by object class, area selection by min/max value, line selection by min/max length, contour line selection by interval, line simplification by Douglas-Peucker, line simplification by Lang, line smoothing by Gauss-filtering and building outline simplification. In addition to these methods of generalisation, the GiMoDig project makes use of preprocessed generalisation operations (that is, the results of generalisation are stored in a multiple representation database (MRDB) structure). The idea of an MRDB is to represent different views on the same physical objects in one data set. These views can stem from different views of the world, different applications, or more commonly different resolutions [208].

Genericity

The GiMoDig project does not aim to support the development of spatially-aware applications. However, the platform was re-used internally by the project to develop a number of mobile applications with map-based interfaces. This reuse was made easier due to the project's adherence to OGC Web-standards.

A major outcome of the project was the results of a user study. During the study it was noticed that users need meaningful map entities that are adapted according to their context of use. The authors suggest that maps need to be available at different scales, and should provide comprehensive information in various formats to various types of devices. These results have contributed to the design of a model for spatial middleware presented in this thesis. Specifically, the requirements for a generic framework to support a range of spatially-aware applications are based on the assumption that map-based interfaces will be highly dynamic, available at multiple scales and need to adapt to user's context (Section 3.2).

2.1.7 M-Spaces

M-Spaces is a spatial model for location-aware services in ubiquitous computing environments developed at the National Institute of Informatics, Tokyo, Japan [200]. Designed to maintain the positions of people, objects, and spaces in the real world, M-Spaces is a general solution that is independent of application-specific services and particular sensors. M-Spaces does not rely on databases and can operate in mobile ubiquitous computing environments. The design of M-Spaces is influenced by three main criticisms of existing spatial models:

- Existing models are constructed in an ad-hoc manner, in that they have been designed for particular sensing systems.
- Existing models have inherently focused on particular application-specific services, for example, user navigation, visualising locations on maps and providing information relevant to the user's current location.
- Existing models cannot be used in ubiquitous computing environments (characterised by resource constrained mobile devices dynamically connected to and occasionally disconnected from wireless networks, often organised in an ad-hoc and peer-to-peer manner) because they rely on access to a centralised infrastructure.

M-Spaces uses a containment relationship model to manage physical spaces identified by their symbolic name. Physical spaces are often organised in a containment relationship, in that each space is composed of more than one sub-space. For example, each floor is contained within at most one building and each room is contained within at most one floor. This model is mapped onto an object-oriented hierarchy of mobile agents.

Availability

The agent-based approach of M-Spaces, as in CRUMPET, allows the location model to remain available to applications despite periodic disconnections from infrastructure-based wireless networks and the limitations of P2P networking. This is achieved by allowing agents migrate between hosts in the environment. However, peer-to-peer collaboration alone does not solve the problem of unreliable wireless network infrastructure. To provide a dynamic model of the users' environment requires a spatial model including meaningful map entities, such as streets, buildings and dynamic objects, in an interpretable format and at multiple levels of detail on hand-held mobile devices. Without this environment model, common application-level spatial services can not be provided to spatially-aware applications. The model for spatial middleware proposed by the current thesis shares M-Space's P2P networking features but uses this capability to disseminate non-topological geometry-based spatial data.

Limitations of Mobile Devices

The proxy components in M-Spaces maintain portions of the tree on behalf of other hosts. This allows a less capable mobile device to access services based on their location in the environment without having to maintain a portion of the environment model themselves. However this approach introduces the possibility that an unreliable or low bandwidth wireless link between a proxy and mobile client could severely degrade the user experience. In addition the use of proxy services is not with the goal of minimising energy consumption on hand-held mobile devices and could potentially consume more energy than performing spatial operations locally.

Spatial Operations

Unlike other systems analysed in this section (Deep Map, GiMoDig, Lol@), M-Spaces does not base its spatial information model on geographical coordinates. Instead, it is based on geographical containment between objects. As coordinates are not involved, there is little GIS technology used. Spatial containment and geocoding symbolic names are the only spatial operations built into the framework.

At the core of M-Spaces is the notion of connecting virtual services and the real world. These virtual services can be any application specific service including, in the case of spatially-aware applications, spatial services. For example, M-Spaces has been used to develop 'Follow Me' applications that allow the user interface to a persons applications to follow them from computer to computer, PDAs have been used to control nearby electric lights and applications have been developed that display maps of the user's surroundings [201].

Genericity

In contrast to projects such as Nexus, M-Spaces is designed primarily to address spatial models of indoor environments, supporting mobile users from stationary computers distributed in a smart environment.

The lack of coordinate-based spatial data limits the available spatial operations and makes M-Spaces unsuitable for providing geometric or topological representations of the user's environment. This limits interoperability with GIS data formats making the reuse of the containment relationship model in different environments difficult.

2.1.8 GinisMobile

GinisMobile is a mobile GIS application framework that includes support for management and presentation of raster and vector spatial data, as well as dynamic data about mobile objects [183, 182]. Developed at the Computer Graphics and GIS Laboratory, University of Nis, Serbia and Montenegro, GinisMobile is designed to act as a platform on top of which developers may experiment with context-aware applications. GinisMobile provides mobile applications with spatial data in the form of raster maps and vector map objects in addition to supporting spatial operations via a server running a GIS.

Figure 2.8 illustrates the architecture of GinisMobile. A client/server architectural approach is taken with the client responsible for displaying static spatial layers (rasterised to a single map layer) and dynamic map components (for example, moving

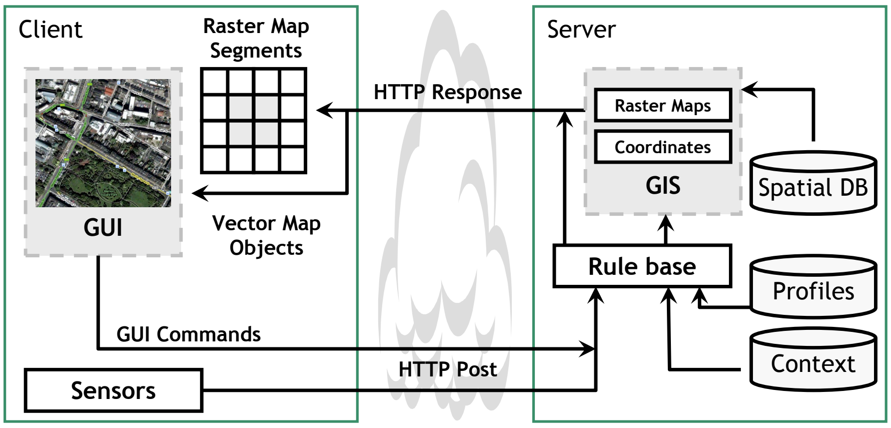


Figure 2.8: GinisMobile architecture (Source: Predic et al. [183])

objects). Static objects are presented on the background map, while moving objects are superimposed over the background. Since raster segments are static in nature and change rarely, a local cache of frequently used segments is maintained. The server contains all the spatial information in the system and is responsible for spatial data management, querying, analysis and presentation over the wired and wireless Web.

Availability

GinisMobile relies on a network connection to deliver services. Changes in the user's environment (detected by sensors) and user input are both communicated to the server, which responds with an updated interface. This design decision was taken to minimise the processing requirements on the mobile device but has the result of making any applications built using the GinisMobile framework unusable in the presence of limited bandwidth or unreliable connectivity.

Limitations of Mobile Devices

GinisMobile takes into account the limitation of mobile devices both by minimising processing on the client side and through context-awareness, allowing the server to adjust the set of offered services according to technical characteristics of the user's environment (processing power, memory, display characteristics and network connection).

GinisMobile makes use of two novel techniques for supporting spatially-aware mobile applications. Firstly, the map tiling algorithm, which caches raster map tiles on the client and discards them using an LRU algorithm, is a novel feature that successfully reduces the dependence on reliable network connectivity and minimises memory requirements. Secondly, like GiMoDig, vector map layers on the client, implemented using SVG, enables increased interactivity and usability by supporting the rendering of dynamic map objects without overly burdening the mobile device with computation.

Spatial Operations

All spatial information in GinisMobile exists in geospatial databases on the server-side. The spatial data is accessed and manipulated using services implementing standard OpenGIS web services (WMS, WFS, etc.), as well as with core LBS services defined in OpenLS service framework, namely: Geocoder, Reverse Geocoder, Directory, Route, Navigation, Gateway and Mobile Presentation services [106]. This rich set of spatial services for mobile applications demonstrates the potential of GIS technology to support spatially-aware mobile applications. The model for spatial middleware presented in this thesis provides many of these services.

Genericity

The goal of GinisMobile is to create a framework that is "capable of supporting development of GIS applications in the mobile environment that include integration, management, querying, analysis and visualisation of geospatial data" [182]. Similar to Crumpet and GiMoDig, GinisMobile heavily exploits web standards to provide this support. The framework itself is shown to be extensible through the development of a number of applications [182].

2.1.9 Point to Discover

Point-to-Discover (P2D)[4] is a research project from the Telecommunications Research Centre, Vienna, Austria [84, 85, 213]. The P2D project built an infrastructure platform to support spatially-aware mobile applications. The P2D platform is server based and has a 2.5D environment model (each object is represented by a two-dimensional footprint polygon extruded by a single height value). It is designed to provide mobile device

[4]http://p2d.ftw.at/

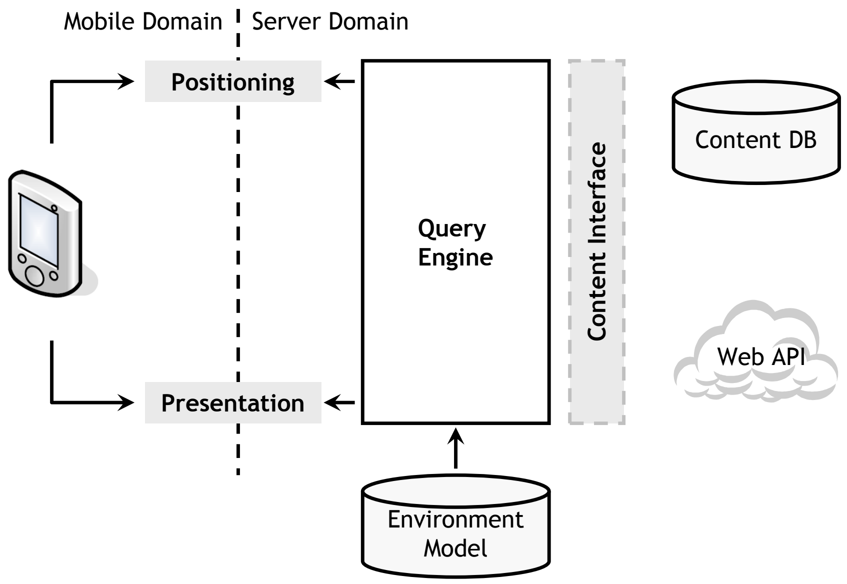


Figure 2.9: Point to Discover application framework architecture

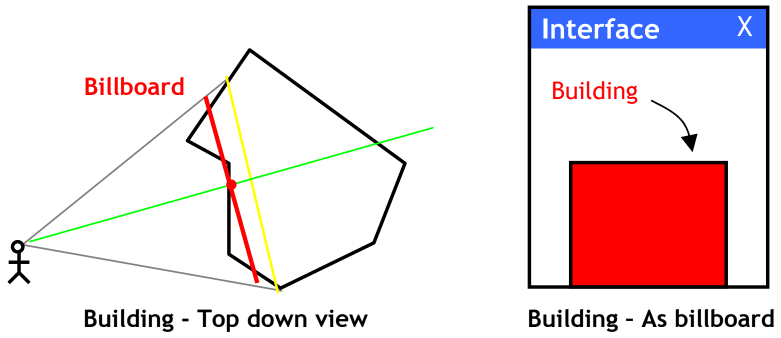


Figure 2.10: P2D LVis billboard principle

developers with the building blocks to prototype innovative interaction metaphors such as 'Smart Compasses' pointing in the direction of interesting places, 'Smart Horizons' allowing users to look beyond their real-world field of view or 'Geo-Wands', virtual geographic pointers for the selection of surrounding objects and attached services.

The P2D project takes a web-based architectural approach to supporting spatially-aware mobile applications. P2D is designed to run on PDAs and mobile phones that have positioning, orientation and tilt sensors. The information from these sensors is delivered to a server-side geospatial query engine. The query engine selects content based on its visibility from the user's location and returns it to the client (Figure 2.9).

The Local Visibility Model (LVis) consists of spatial modeling metaphors to describe the geometrical arrangement of the search result space: Points of Interest (POI) and Billboards. POIs are point-shaped references to geocoded content, the result of a traditional geospatial query. Billboards are a method of including the actual geometry (a simplified three-dimensional geometrical representation of a 3D space) of the environment in the query result. Figure 2.10 illustrates how LVis can be thought of as a "cardboard cutout" version of the search space. The facades of buildings visible from the user's location are determined and included in the search result.

The P2D project sees two advantages to the LVis approach:

1. Local knowledge of the environment's geometry on the mobile device makes advanced sensor driven interfaces possible. (For example, the user interface illustrated in Figure 2.10 presents the spatial query result in a schematised panorama view of the environment.)

2. It is argued that locally stored geometry could enable real-time user interfaces. For example, if a mobile client had spatial data describing the three-dimensional structure of its environment, it could react to changes from sensors without the need to re-query the server.

Availability

P2D applications store fragments of the environment geometry locally to limit the application's reliance on always-on connectivity. However, the environmental information that is cached is specific to the user's location and once this changes the server must be contacted to update the mobile device's environment model. The P2D project asserts that mobile applications should not depend on always-on connectivity, particularly in emergency situations. P2D suggests that synchronisation should occur as soon as connectivity with the network is re-established, or it can be performed in a decentralised manner, by exchanging spatial knowledge stored on one device with other devices. Although a distributed approach is suggested, the applications developed using P2D have all been client/server.

Limitations of Mobile Devices

The limitations of mobile devices are taken into account in the design of the Local Visibility Model (LVis). LVis is an XML data format that encodes geographical knowledge about the environment based on their visibility. Mobile applications periodically retrieve a new LVis from the server and update the user interface accordingly. Device limitations are further taken into account by the architectural design decision to place all the spatial data and spatial operations on the server-side.

The lack of geometric spatial information in the LVis model coupled with the design decision to perform all spatial operations on the server-side prevents spatially-aware mobile applications from maintaining a model of the users environment suitable for supporting highly interactive mobile applications. This results in mobile applications that are dependent on reliable wireless networks coverage, which is often not the case in the real world.

Spatial Operations

The P2D framework incorporates spatial query operations based on visibility and field of view, a 2.5D environment model, and a presentation independent data exchange format for geospatial query results. P2D does not support map-based interfaces for mobile devices. Instead, the concepts of field of view and visibility are exploited to support information discovery in mobile applications. This design decision has the consequence of limiting the amount of work that the mobile device needs to do and limits the volume of data that must be exchanged between client and server. However, the lack of any real world geometry on the mobile client limits the device's ability to produce a map and makes it impossible to support navigation.

Genericity

To keep the barrier of entry low for developers previously not involved in geospatial application development, the LVis uses simple spatial modeling metaphors to describe the geometrical arrangement of the search result space. The P2D project also makes a number of contributions to knowledge in the area of spatially-aware mobile applications through user studies:

- P2D demonstrated that map-based interfaces, while still ranked among the most popular forms of user interfaces, are not the only options for usable and intuitive interfaces to spatial information.
- P2D was the first project to evaluate the potential value to users of an explicit indication of the visibility of nearby geographic features and points of interest.

	Cyberguide	Deep Map	CRUMPET	Nexus	Lol@	GiMoDig	M-Spaces	GinisMobile	Point-to-Discover
Architecture									
Client/Server		●			●	●		●	●
Locally Centralised	●								
Distributed			●	●			●		
Spatial Operations									
Selection	●	●	●	●		●		●	
Proximity		●	●	●		●	●		
Map Production	●		●	●	●	●		●	
Levels of Detail					●	●			
Visibility			●						●
Generalisation		●				●			
Navigation	●	●		●	●			●	
Availability	●				●		●	●	●
Mobile Device Limitations	●		●			●	●		
Genericity									
Interoperability			●	●		●		●	●
Reuse	●	●					●		

Table 2.1: Spatial services for mobile applications

- P2D users found orientation awareness a useful feature of mobile spatially-aware applications.

These contributions have informed the design and contributed to the requirements for a reusable generic framework for spatial services presented in this thesis.

2.1.10 Summary

This section has presented the state of the art in spatial services for mobile computing applications. Table 2.1 summarises the features of the surveyed systems.

The architecture of a system, while not being directly apparent to the user, has a serious impact on the system in terms of extensibility, adaptability and reuse. The majority of these systems take an infrastructure-based approach, either client/server or distributed. Only the Cyberguide project localised all spatial functionality on to the mobile device and that was largely due to the lack of a suitable wireless network deployment at the time. A common feature of a number of the systems is that they are multi-tiered and are increasingly incorporating GIS servers as a component of the infrastructure (Deep Map, Lol@, GiMoDig). The predominantly client/server architectures prevent these systems from maintaining a model of the user's environment that could support spatial services without relying on a reliable network connection.

The most common spatial operations supported by these systems are selection, proximity and map production. Selection is the querying or retrieval of spatial data for a specified region. Proximity is the distance-based retrieval of information or triggering of events and map production is the creating of a cartographic map in either vector (GiMoDig) or raster format (Cyberguide, CRUMPET, Nexus, Lol@). Navigation support including route generation and wayfinding is a common feature of this class of system, possibly due to the majority of these projects targeting the tourism domain (Deep Map, CRUMPET, Lol@, GinisMobile). Increasingly, these systems are producing maps at various levels of resolution and detail. The P2D project and M-Spaces are the only surveyed systems that do not assume that a map-based interface is all that is required on the mobile device. P2D enables developers of spatially-aware applications to experiment with new interaction metaphors and to prototype user interfaces that offer experiences beyond what is offered by today's applications. However, even P2D did not communicate spatial data to mobile devices, making it impossible for devices themselves to maintain an environment model that could be used to adapt behaviour. The M-Spaces project targeted smart environments where the lack of client side spatial services is less of a limitation as ubiquitous wireless network coverage can be provided more easily.

Only three of the reviewed systems (Cyberguide, Lol@, P2D) provide spatially-aware applications with services that are not dependent on uninterrupted wireless network coverage. Cyberguide achieves this as a side effect of the locally centralised architectural approach. In contrast, the Lol@ project explicitly acknowledges the fact the mobile applications must degrade gracefully in the face of limited bandwidth or loss of network connection. To account for this, Lol@ performs some functions on the mobile device, reducing latency, improving robustness and conserving energy usage. Similarly, P2D cache's data on the mobile device allowing for continued operation despite periodic network disconnections. The Deep Map project advocated the approach of caching spatial data on the mobile device but noted that lightweight variants of spatial operations would be required on mobile devices to exploit spatial data to provide useful spatial services.

These projects take the limitations of mobile devices in terms of computation, memory and energy into account in one of three ways: 1) Computationally complex operations are offloaded to more capable servers (CRUMPET, Nexus, GiMoDig); 2) Generalisation [127] is used to reduce the complexity of spatial data to a point where it may be managed by a mobile device (GiMoDig); 3) The limitations of the target platform are accounted for statically at design time by implementing minimal functionality or tailoring spatial data to meet the capabilities of a particular platform (Cyberguide, Lol@, M-Spaces, P2D). None of the surveyed systems cited energy usage as a device limitation they were concerned with. This is despite the client/server architecture, which requires significant use of energy consuming wireless networking, being the most common architectural approach to supporting mobile spatially-aware applications.

The systems presented in this section provide spatial services to mobile, spatially-aware applications. However, they provide varying support to the developers of these applications. Two main approaches to enabling the development of spatially-aware applications can be identified; Reuse and Interoperability. Systems that support development through reuse, do so through the provision of a framework, service or platform (Cyberguide, Deep Map, Nexus, M-Spaces and GinisMobile). Alternatively, interoperability through the adoption of web standards or definition of open XML data formats facilitate the development for applications that can directly interact with these systems (GiMoDig, P2D).

2.2 Commercial GIS & Web Mapping

The source of spatial data in many of the systems reviewed in this chapter is a GIS. There are a small set of general purpose GIS available, the majority of which are commercial applications. The Intergraph Corporation's *MGE/MGA* system, Pitney Bowes Software *MapInfo* platform and ESRI's[5] *ArcGIS* are general purpose systems that meet the needs of various application domains. Particularly relevant to this thesis

[5]Environmental Systems Research Institute

is the ArcGIS Mobile SDK [70] product from ESRI, as it deals with the use of spatial information in mobile environments. This section will review the capabilities of this commercial development platform and application framework using the same criteria with which we analysed spatial services for mobile computing (that is, Availability, Limitations of Mobile Devices, Spatial Operations, and Genericity).

In addition to commercial GIS, there is related work in the area of web mapping. In recent years there has been an explosion of mapping applications on the web, such as Google Maps [91], Multimap [156], Microsoft Live Search Maps [145], and Yahoo! Maps [241]. The addition of API interfaces to these web services and distribution of free mobile clients make these systems a possible means of accessing spatial information on hand-held mobile devices.

2.2.1 ArcGIS Mobile SDK

We have previously defined GIS as a system capable of capturing, storing, analysing, and displaying graphically referenced information. However, modern GIS go beyond this by providing a platform on which enterprise scale domain specific applications can be developed and deployed. ArcGIS Mobile is an example of a mobile GIS platform that provides access to GIS services over wireless networks to a range of mobile devices [70]. ArcGIS Mobile provides mobile-based GIS functionality that includes mapping, spatial query, sketching, GPS integration, editing, and wireless data access to ArcGIS Server services.

ArcGIS Mobile is designed to support field data collection, essentially providing an extension to the desktop-based GIS in a mobile environment. The goal of ArcGIS mobile is to allow spatial data to be annotated and for these annotations to be made available to a fully featured desktop-based GIS on return to the office.

Figure 2.11 illustrates the architectural approach taken by ArcGIS Mobile. In this model, mobile applications are designed on the desktop and packaged along with a database of spatial data before being deployed to the mobile device. The design and implemention of mobile applications is supported by the ArcGIS Mobile SDK, a suite of .NET components for developing custom server-centric lightweight applications. Mobile devices communicate with a GIS server for the purpose of transferring spatial data. Once spatial data is on the device, the application allows users to view,

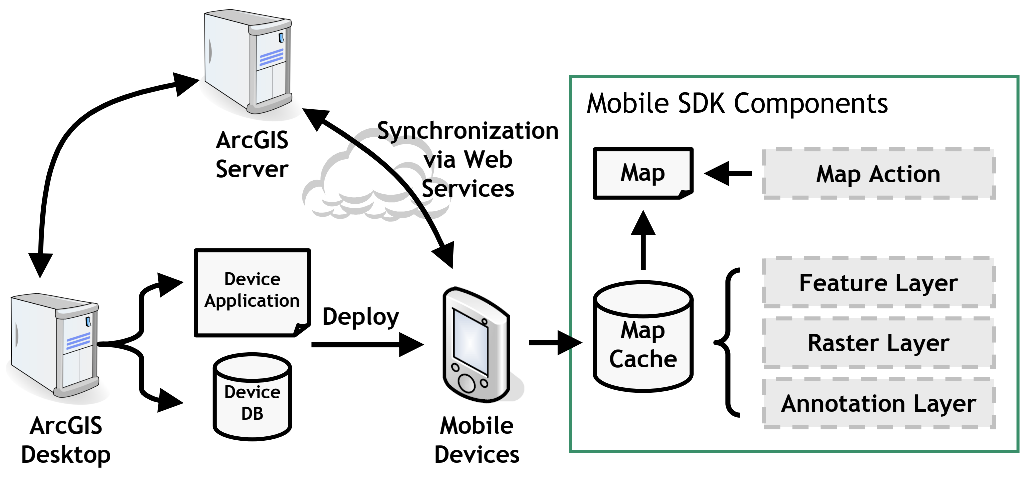


Figure 2.11: ArcGIS Mobile SDK architecture (Source: ArcGIS Mobile 9.3 [70])

navigate, collect and update maps and GIS features.

2.2.1.1 Availability

ArcGIS Mobile applications work in both connected and disconnected environments. This is achieved by caching data on the mobile device (Guide [44], GinisMobile [183]) to allow the user to perform their task when there is no connection to the server. The detail of the spatial data and the real world extent that it represents is limited by the memory of the mobile device. A connection to the server is required to synchronise updates [221]. The caching approach goes some of the way to mitigating the challenges of maintaining an environment model without relying on a wireless network (RQ-1) but is not a solution. Synchronisation is still required, spatial models are static and spatial information is available at a single scale.

2.2.1.2 Limitations of Mobile Devices

The ArcGIS Mobile SDK is designed with mobile devices in mind. The spatial data on the mobile device is authored specifically for mobile use and does not contain the same level of detail as its desktop equivalent. This loss of information is acceptable as the mobile application is designed to operate in tandem with a more capable desktop GIS that supports more detailed spatial data editing on return to the office.

The developers of the ArcGIS Mobile SDK made map rendering performance a key requirement. This is supported by the small 540kb footprint of the SDK libraries. However, ArcGIS Mobile is not a generic tool for spatially-aware applications. It is designed to support a single class of mobile application and does not provide many of the spatial services typical of spatially-aware applications (for example, route generation).

2.2.1.3 Spatial Operations

ArcGIS Mobile is not a middleware product and so provides limited ability to manipulate spatial data on the mobile device. The spatial operations supported relate to viewing and editing spatial data attributes. These include:

- View and navigate mobile maps.
- Collect, edit, and update new and existing GIS data.
- Search for and manage a list of GIS features.

This limited set of spatial operations prevents the use of ArcGIS mobile as a generic framework for developing spatially-aware mobile applications.

2.2.1.4 Genericity

ArcGIS Mobile includes an SDK that supports the development of one type of spatially-aware mobile application (that is, field data collection). The SDK does not feature a set of spatial operations suitable for supporting the interpretation of a model of the users environment for the purpose of adapting application behaviour or interface. The SDK is also tied to particular proprietary server platform and requires at least intermittent connectivity with an instance of this GIS server. However, ArcGIS Mobile does make use of standardised geographic data formats and communication protocols.

2.2.2 Web Mapping

Web mapping is the process of generating and delivering maps on the World Wide Web [149]. Web mapping is primarily a dissemination medium of spatial data from traditional GIS. Examples of web mapping applications include Google Maps [91], Multimap [156], Microsoft Live Search Maps [145], and Yahoo Maps [241]. These applications typically offer street maps, searches and aerial/satellite imagery. These services are based on raster tiles organized in a quad tree scheme, that are loaded asynchronously with XMLHttpRequests.

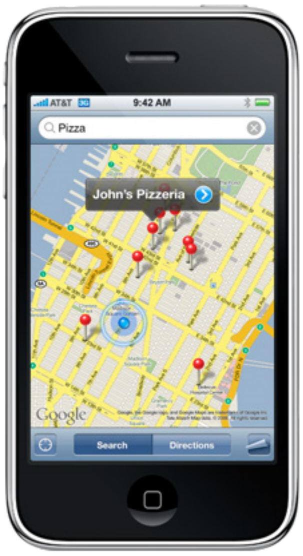


Figure 2.12: Google Maps for the iPhone 3G (Source: Apple.com [12])

In recent years, web mapping services have begun to adopt features more common in GIS. For example, web maps are increasingly providing spatial services such as geocoding and route generation. Services such as Google Maps, Multimap and OpenLayers, expose an API that enable users to create custom applications while Google Maps and Live Maps allow users to annotate maps and share them with others.

2.2.2.1 Google Maps

As an example of the use of web mapping on mobile devices, we look at the Google Maps for Mobile (GMM) application that is distributed for free with the iPhone 3G (iPhone OS 2.0.2, August 18th 2008). GMM, like its desktop counterpart, provides draggable maps, driving directions, local business listings and satellite imagery. GMM also provides mobile phone specific features including:

- Your current location (Figure 2.12). Your current location is shown on the map using built-in GPS, or a Bluetooth GPS sensor or using WiFi or mobile phone towers within range.
- Business listings. Search for businesses by name (for example, "John's Pizzeria"), or by type (for example, "pizza").
- Turn-by-turn driving directions.

- Real-time traffic data. Roads are colored green, yellow or red, based on real-time traffic data.
- Favourites. Bookmark your favourite places so that you can easily return to them on the map

2.2.2.2 Multimap Open API

The Multimap Open API [156] from a London based company (acquired by Microsoft in December 2007) is another example of web mapping. Unlike Google Maps, Multimap does not offer a mobile specific application but does include two programmatic interfaces. The first of these interfaces is a JavaScript API designed for building browser-based AJAX applications with features such as draggable maps, overlays, polylines, map markers and information boxes. The other interface is an XML-based web services interface that is accessed over the Internet, and supports geocoding, routing, coordinate transformation and static maps. Either of these interfaces could be exploited to provide map-based services to hand-held mobile devices.

However, neither of these examples (Google Maps and Multimap) address the challenges of providing an architectural model that is robust enough to support constant application availability in a mobile wireless networking environment. In addition, these examples are not designed with the limitations of hand-held mobile devices in mind. Specifically, web mapping services such as these have been criticised for being merely down-sized versions of their large-screen counterparts that shrink the same user interface onto the tiny mobile device [212]. Empirical research has shown that it is inappropriate to apply desktop idioms to mobile user interfaces [122]: mobile users are typically occupied with real world tasks, and interactions are driven by the external environment [175].

2.3 Chapter Summary

This chapter has reviewed and analysed spatial services for mobile computing, which have been presented and compared based on: their architectural design choices; the spatial operations they perform; their provisions for maintaining availability in the face of unreliable wireless networks; their approach to managing the limited resources of mobile devices; and the contribution they make to the development of further spatially-aware applications.

Of the spatially-aware applications presented, very few were capable of executing on a hand-held mobile device. The range of services provided by spatially-aware applications includes navigation, communication, maps and location specific information. The most common architectural approach to providing these services is client/server. The systems that adopted a different architectural approach, did so by storing spatial data locally. However, these systems are confined to limited areas and are difficult to update.

Experiences in evaluating these applications highlighted the need to solve challenges posed by the nature of wireless networking [178, 222, 213]. Currently, the effect of unreliable communications is addressed by caching spatial data on the mobile device. This strategy is limited by the memory available on the mobile device and limits the applications ability to remain up to date. The caching approach also raised the problem of how to produce a dynamic map-based interface from spatial data in a mobile environment, which is a task that consumes significant resources.

The spatial services for mobile computing that have been presented typically took the form of application frameworks and web services. These systems were based on client/server architectures with the server-side often incorporating a traditional GIS server. An increasing trend towards supporting open standards to allow the interoperation of GIS components is evident. In addition, generalisation operations have begun to be used as a means of adapting spatial data for use on mobile devices. However, these operations remain confined to servers.

In addition, none of the projects reviewed here have considered minimising the consumption of energy on mobile devices a design goal. There is an energy trade-off in performing computationally intensive tasks (for example, spatial data manipulation) locally versus communicating these tasks to more capable servers. Although some projects made use of servers to perform these complex tasks [120, 200, 168], none analysed the energy consumption trade-off.

Commercial GIS systems are also incorporating the ability to collect and edit spatial mobile using mobile devices in this field. These systems can operate in a disconnected environment, but feature spatial operations that are limited to viewing and editing spatial data attributes and features. Similarly, web mapping approaches provide a means of accessing spatial data in mobile environments. However, web mapping relies on a reliable network connection, provides no environmental model and access to a very limited set of spatial services. The interface to web mapping applications is in raster format, precluding any ability to adapt the interface beyond what is available on a desktop-based web browser.

The major criticisms and shortcomings of the systems presented in this chapter can be summarised as:

- Existing systems do not provide spatially-aware mobile applications with the ability to maintain a dynamic model of the user's environment suitable for supporting highly interactive mobile applications. Where models are maintained, they are provided as infrastructural services to mobile applications and require a reliable network connection to maintain the user experience.
- Due to the predominantly client/server architectures, these systems do not include algorithms that can manage the complexity and volume of spatial data in real-time on a hand-held mobile device.
- Without access to spatial data or spatial operations on mobile devices, existing models can not provide reliable access to common spatial services such as map rendering and route generation.
- To date, minimising energy consumption on hand-held mobile devices has not been a design goal of spatially-aware applications. Existing systems do not take into account the energy trade-off in performing computation locally versus communicating tasks to more capable servers.
- There is a lack of reusable software for developing spatially-aware mobile applications. Existing frameworks are server-centric or support the development of software in only one application domain. This requires developers of spatially-aware applications to repeatedly tackle the challenges of mobility, device resource limitations and unreliable wireless networks. Generic reusable implementations of spatial services are needed that do not overburden the limited resources of mobile devices and are not dependent on continuous network connectivity.

The next chapter presents the design of a middleware for spatially-aware mobile applications including an architectural model and a set of algorithms designed to minimise the processing time and power consumed on hand-held mobile devices while providing uninterrupted access to common spatially-aware application services.

Chapter 3

Design

Existing spatially-aware applications do not maintain a dynamic model of the user's environment that is suitable for supporting highly interactive mobile applications. Where static models are maintained, they are provided as infrastructural services, requiring a reliable network connection. Generic reusable implementations of spatial services are needed that do not overburden the limited resources of mobile devices and are not dependent on network connectivity.

This chapter describes the design of a model, algorithms and framework for mobile spatially-aware applications. The model for mobile spatial middleware described in the current chapter provides spatially-aware applications with the ability to maintain a dynamic model of the user's environment by exploiting the ad-hoc collaboration features of the Hermes framework (RQ-1). The algorithms designed to provide spatial operations on hand-held mobile devices (RQ-2) include a multiple representation database, a hierarchical spatial index and an on-demand model-oriented generalisation process. The spatial middleware services, (RQ-3) including adaptable map rendering, spatial reasoning, coordinate transformation, route generation and visibility determination are also presented. These services minimise the processing required by using clipping, buffering and caching. Finally, the design of a generic framework for spatial services that incorporates the algorithms and services of this thesis into a reusable library of spatial tools is presented (RQ-5). This framework supports the development of spatially-aware applications that do not depend on continuous, reliable wireless networking, do not overburden the limited resources of mobile devices, and maximise the battery-life of hand-held mobile devices. Figure 3.1 maps the research questions from

	Model for Spatial Middleware (Section 3.1)	Multiple Representation Database (Section 3.2.1)	Spatial Index (Section 3.2.2)	Generalisation (Section 3.2.3)	Adaptable Map Rendering (Section 3.3.2)	Spatial Reasoning (Section 3.3.3)	Coordinate Transformation (Section 3.3.4)	Route Generation (Section 3.3.5)	Visibility Determination (Section 3.3.6)	Generic Framework (Section 3.4)
		Algorithms			Spatial Services					
RQ-1	●	●								
RQ-2		●	●	●						
RQ-3				●	●	●	●	●	●	
RQ-4	See Energy Evaluation Design (Section 5.2)									
RQ-5										●

Figure 3.1: Mapping research questions to design elements

Section 1.2 to the design elements from this chapter.

Throughout the design process there were two overriding concerns. Firstly, the design had to be implementable on hand-held devices and be efficient enough not to overburden their limited memory and processing resources. The design is heavily optimised towards achieving a lightweight implementation. The design minimises the computational complexity of algorithms so that they will consume less energy when executed on mobile devices. Secondly, the design has to support ease of reuse by using logically decomposed components that separate different application concerns and reuse design metaphors familiar to developers of mobile and GIS applications. These techniques contribute to the extensibility of the resulting framework, ultimately reducing the cost of development of spatially-aware applications.

3.1 Model for Mobile Spatial Middleware

Spatially-aware mobile applications require a dynamic model of the user's environment that must contain meaningful map entities such as, streets, buildings and dynamic objects. This data must be in an interpretable format and be accessible despite the inherent unreliability of wireless networks, mobility of users and limited nature of hand-held mobile devices.

Existing spatially-aware applications take a client/server approach, where a server maintains a model of the environment and distributes spatial data from this model to mobile clients (cf., Guide [43], comMotion [136], GeoNotes [68] and REAL [19]). Of the systems reviewed in Chapter 2, only the Cyberguide project [129] was designed to localise spatial data on mobile devices. That system is, however, limited by its raster-based spatial data model (Section 1.1.3).

To meet the demands of a dynamic, interactive environment without compromising the user experience of spatially-aware applications, spatial middleware services must have constant access to an environment model. Environment models are populated with spatial data but it is not feasible to store all the spatial data an application will need ahead of time due to the quantity of data, device storage limitations and because the spatial data changes over time. Section 3.1 presents the system model assumptions in terms of the hardware environment and existing middleware support for mobility.

Based on these assumptions, a middleware approach to maintaining a model of the user's environment is described.

3.1.1 System Model Assumptions

The model for spatial middleware presented in the current thesis makes a number of architectural design decisions based on technical assumptions in the areas of: 1) the computing environment; and 2) the existence of middleware for mobile computing.

3.1.1.1 Computing Environment

We assume an asynchronous distributed system where heterogeneous, distributed mobile devices communicate using various wireless networking technologies with limited range. Mobile devices can experience unanticipated and possibly prolonged disconnections; routinely change their point of network connection, potentially resulting in a transition from one network infrastructure to another; and are sometimes organised in an ad-hoc and peer-to-peer manner. The computing environment is viewed as collaborative, with mobile devices capable of discovering and serendipitously communicating with peers and also exchanging data with fixed infrastructure. Application logic is pushed to the edge of the network (that is, the logic resides entirely on the mobile devices).

3.1.1.2 Middleware for Mobile Computing

The Hermes[1] project at Trinity College Dublin is the umbrella project under which the work in the current thesis was completed. The initial requirement for a spatial middleware for mobile applications was drawn from the experience of this project and for that reason the model presented in this thesis is designed with the assumption that the Hermes mobile application framework provides services for on-demand collaboration between heterogeneous mobile devices.

Figure 3.2 illustrates the Hermes framework architecture. The current thesis is concerned specifically with three of the services for collaborative mobile applications that this application framework provides: *Communication*, which includes *Service Discovery*; *Collaboration*; and *Context Modeling*. The *Collaboration* and *Communication*

[1] `http://www.hermes.dsg.cs.tcd.ie/`

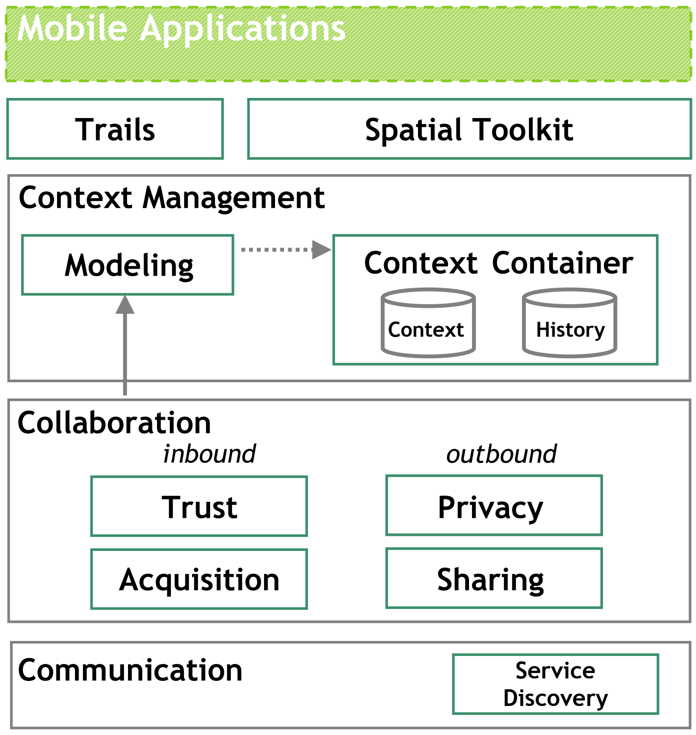


Figure 3.2: The Hermes mobile application framework architecture

layers acquire and disseminate spatial data in a ubiquitous computing environment. Specifically, the *Communication* layer is responsible for managing communication with infrastructure and peer devices over a variety of networks and discovery of other devices supporting the Hermes framework.

The *Collaboration* component divides its responsibilities between inbound and outbound considerations. On the inbound side (left) are the *Acquisition* and *Trust* components. The *Acquisition* component is responsible for proactively acquiring data on behalf of an application. On the outbound side (right) are the *Sharing* and *Privacy* components, responsible for exchanging data with peers.

The *Context Modeling* layer provides position and orientation information that can be used to adapt spatial data visualisation. The following section describes how the dissemination of spatial data is achieved using the collaboration services of the Hermes mobile application framework.

3.1.2 Spatial Data Dissemination

We have assumed a computing environment characterised by mobile devices serendipitously communicating with peers and the existence of middleware services for collaborative mobile applications. In spatially-aware applications, both the physical structure of the user's environment and the relationships between real world objects are modeled as spatial data [157]. Based on the assumptions we have made, it is possible to partition spatial data into fragments representing real world regions and collaboratively distribute these fragments throughout the user's environment. Mobile devices may acquire these fragments as they move through the world and use them to augment and extend their own environment model.

Transferring raw spatial data (geometry and attributes) to mobile devices rather than making derived services available via a wireless network was first proposed by the Cyberguide project [129]. Cyberguide found that locating spatial data (raster maps in Cyberguide's case) on the mobile device allowed for constant availability of applications but limited the ability to update the spatial data. Other projects overcame this limitation at the cost of wireless network dependence [25, 14].

Some projects took an entirely different approach to providing spatial services in ubiquitous computing environments. M-spaces developed a spatial model that was not based on traditional spatial data. This approach limited the ability to inter-operate with other mobile applications, thus causing difficulties in re-using existing spatial data produced by GIS applications.

Other projects seeded the environment with infrastructure serving spatial data for a region [161]. However these projects used proprietary spatial data models and formats to transfer limited types of spatial data. The overhead in interpreting these formats on mobile devices limits their ability to be reused by a generic range of spatially-aware mobile applications.

Figure 3.3 illustrates how the Hermes framework supports collaborative spatial data dissemination. The environment is made up of mobile devices and fixed infrastructure. All of these elements execute the lower layers of the Hermes framework (that is, they can all participate in serendipitous collaboration).

The process of acquiring spatial data for a region begins with an ad hoc device discovery process. Each collaborating node broadcasts a beacon advertising its network address and listens for replies (Figure 3.3, 1). Any device within range that hears an advertisement from a device it hasn't seen before will reply with a message containing a list of the available spatial data (Figure 3.3, 2). The device then makes a decision as

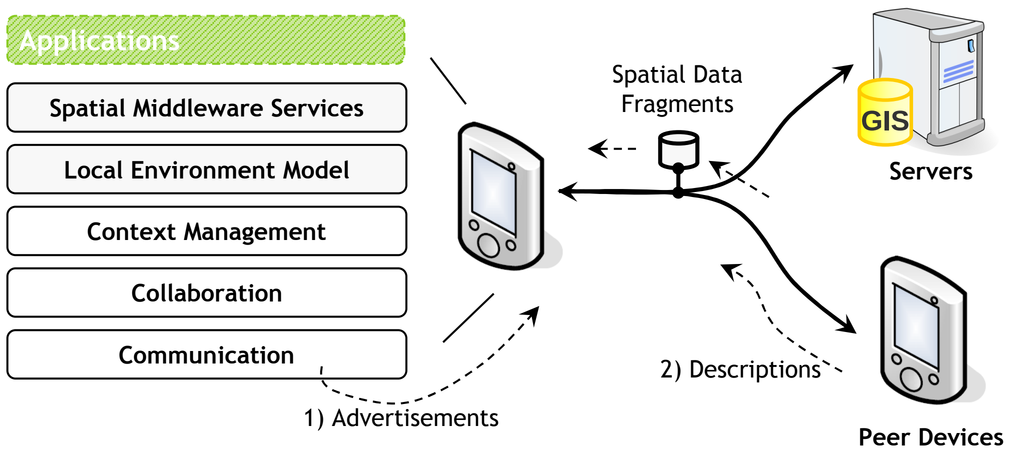


Figure 3.3: Architectural model for mobile spatial middleware

to whether it needs spatial data describing a particular region and acquires it. Once acquired, the new fragment of spatial data is added to the devices' advertised list of available data to be shared with neighboring devices.

The advantages of this approach are that through serendipitous communication, a spatial middleware can acquire only the spatial data it needs, and store it locally on the device. No interaction is required with the user, and applications do not need to be aware of this process. The distributed approach avoids server connectivity issues, potential network bottlenecks and satisfies the need for timely access to spatial data.

However there are some limitations to this approach. The model assumes a level of trust between parties taking part in the system so that inaccurate spatial data is not maliciously exchanged. Partitioning spatial data into suitable fragments, distributing the fragments and ultimately re-integrate possibly overlapping and erroneous spatial data on a mobile device is also a challenge [81, 219]. This challenge is addressed at the *Context Management* layer of the Hermes framework, where all data fusion is performed and is outside the scope of the current thesis.

The possibility that spatial data would not be available in an environment that is only sparsely populated with other devices is also a concern. One approach to this challenge, and one which our model supports, is to provide high-level spatial data covering a large area and make finer grained detail (that is, the interior of buildings) discoverable as the user navigates the real world. Another approach to supporting information dissemination in sparsely populated ad-hoc networks was taken by Trinity College Dublin in the WAND project [234]. The WAND project seeded a ubiquitous computing environments with infrastructure to increase the availability of nodes.

The model for spatial middleware proposed and reported in the current thesis is based on data dissemination and acquisition functionality common in mobile context-aware middleware. The design allows spatial data to be unobtrusively transferred to a hand-held mobile device. When a mobile application invokes a spatial service such as map rendering, the set of algorithms providing spatial middleware services on the mobile device can perform the necessary spatial operations. This novel approach eliminates the dependency on network connectivity, enabling constant availability of spatial services to mobile spatially-aware applications (RQ-1).

3.2 Local Environment Model

Having acquired spatial data, a model of the user's environment can be maintained with the design of suitable algorithms to support spatial operations on hand-held mobile devices (RQ-2). These algorithms include a multiple representation database that dynamically generates smaller scale representations via model generalisation. This database approximates continuous scale adaptation with stepped levels of detail and supports retrieval and insertions with a hierarchical spatial index.

3.2.1 Multiple Representation Database

The spatial middleware acquires then stores spatial data locally on resource constrained, hand-held mobile devices. To provide a usable dynamic model of the users' environment, it is necessary to make spatial data available in different resolutions and levels of detail to allow for flexible zooming on small displays [79]. Spatial data can automatically be simplified to produce zoomed out overview information from detailed spatial data, but this process is computationally expensive. However, caching the results of this process can minimise the effects of incurring this cost. The current thesis proposes a novel multiple representation database[2] that dynamically generates smaller scale representations on-demand via model-oriented generalisation [208, 181, 77]. The design of the multiple representation database (MRDB) is innovatively based on a standard spatial data model [69], approximates continuous scale adaptation with stepped

[2]Also referred to as *multiple resolution database* in literature [99].

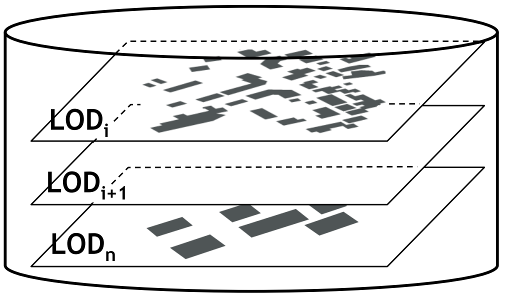


Figure 3.4: Multiple representation database hierarchy

levels of detail [98] and uses a hierarchical spatial index to facilitate insertion and retrieval of spatial objects [188].

An MRDB[3] is a spatial database used to store data at different levels of precision, accuracy and resolution [99, 54]. Figure 3.4 illustrates the representation of spatial objects at different levels of detail. There are a number of design goals for the MRDB. First, the database is designed to provide spatial operations on hand-held mobile devices that are limited in terms of battery power, processing resources and memory (RQ-2). This requires that the data be in an interpretable geometric format (that is, vector representation), on which standard coordinate geometry transformations may be performed. It also requires that the algorithms for retrieving spatial objects from the data store are designed to minimise the computational complexity of locating objects within a defined spatial region. Second, the data store is required to be application-agnostic (that is, it should not assume a particular interaction metaphor but should support a variety of possible solutions) [213]. The data store must also support the storage and retrieval of spatial objects at different resolutions with respect to scale [53, 230]. Each of these scales is referred to as a level-of-detail (LoD) with maps of varying scales reflecting the presentation of spatial objects at different LoDs. A database with multiple LoDs can support the small screens of hand-held mobile devices by reducing the information density in a map interface, providing the ability to zoom in and out on the interface and by providing an internal representation of spatial data with a reduced complexity. Third, LoDs must be produced from dynamic spatial data requiring online model-oriented generalisation (that is, generalisation at the spatial data model level as opposed to at the graphic representation level [53]). Finally, it is

[3]Kilpelainen [116] presents a comprehensive description of MRDBs and develops an MRDB model for generalization of geodatabases of topographic maps.

also a goal to maximise the reusability of existing vector spatial data.

A number of related projects have investigated the use of MRDBs to support mobile applications. The GiMoDig project (Section 2.1.6) used an MRDB structure on the server-side to store the results of preprocessed generalisation operations [208]. Hierarchical topological data structures have been designed to enable progressive transmission of spatial data across networks [28]. This data structure is based on a non-standard data format and includes sequences of generalisation steps that are applied to the coarse data in order to derive LoDs. The generalisation steps themselves are not defined in sufficient detail but are specific to the project, so the data can not be reused by other applications. Han et. al. [100] proposes the use of a hierarchical data structure for multiple representations of vector data sets. Under this suggestion a topological network model is represented at different LoDs. The model facilitates computing adjacency, spatial analysis and efficient storage but can not be used to render map-based interfaces.

There were three possible alternatives considered while designing the MRDB reported in the current thesis: 1) use conventional data structures to represent series of vector maps; 2) use a single data structure with fine grained spatial detail on which cartographic generalisation could be performed; 3) pre-compute LoDs at different predefined scale levels.

The first option, using conventional data structures, would require that a copy of the spatial data be stored redundantly for each LoD. This is the simplest option and provides a workable brute-force solution. However, this approach suffers from issues of storage overhead, and the difficulty of maintaining consistency within the multiple versions of the maps [100, 181]. Alternatively, the second option, storing only a single copy of the spatial data would require that all spatial operations handle complex geometry and all map-based interfaces are rendered using cartographic generalisation, increasing the processing burden and consuming energy. The third option, adopted by the GiMoDig project [230], requires that all spatial data be pre-processed limiting the reusability of existing vector-based data sets and making it difficult to represent the dynamic elements of the user's environment in a spatial model.

This thesis proposes a MRDB spatial model based on the open ESRI Shapefile format [69] that overcomes the limitations of the alternatives considered. The Shape-

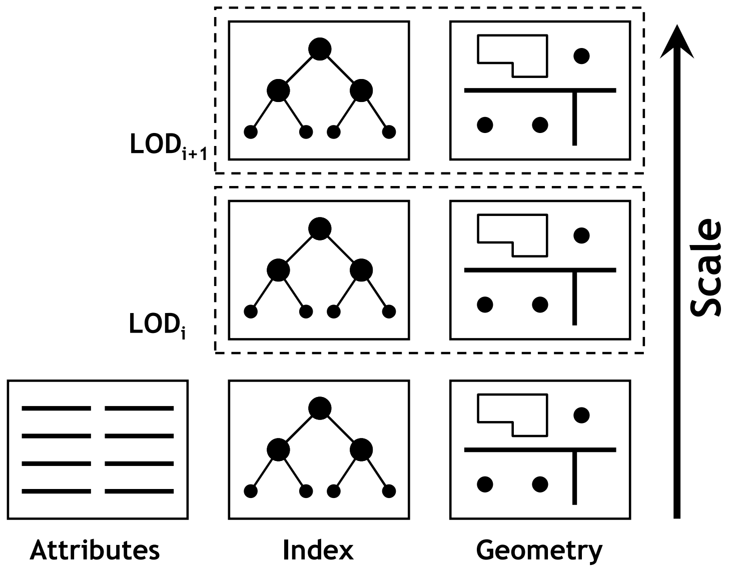


Figure 3.5: MRDB model illustrating Shapefile components

file format supports the storage of attributes, spatial indexes and geometry and is compatible with most commercial GIS tools. The standard technical specification of these files is extended by making use of reserved bytes in the file headers to store scale information. Figure 3.5 illustrates the introduction of additional index and geometry components to the Shapefile format in order to represent spatial data at different LoDs. By choosing an open data format to base an MRDB on, we make it possible to re-use existing spatial data.

Each representation of spatial objects in the MRDB corresponds to a different resolution (or scale). Since each new resolution increases the storage requirements of the data structure, it is desirable to limit the granularity of LoDs. The MRDB partitions scale values into scale ranges, where each range has a single level in the data structure's hierarchy. This achieves more efficient storage and reduces computation at the cost of a loss of continuous adaption while zooming. Although continuous adaption has the potential to improve the usability of map-based interfaces by providing smooth transitions between map scales, this effect can be reproduced using a simple animated transition between static scales [91].

When data is requested at a particular scale, s, the scale range, SR_i, is selected so that $\min(SR_i) < s < \max(SR_i)$ where each scale range, $\{SR_i, SR_{i+1}, \ldots SR_n\}$, has a one-to-one mapping to a LoD (for example, $SR_i = LoD_i$). Once a LoD is determined, spatial data can be read from the data structure at that scale. The algorithm for doing

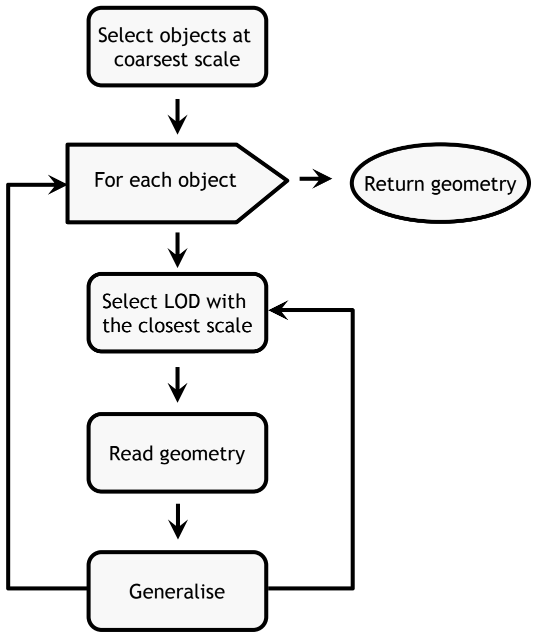

Select objects at coarsest scale
For each object
Return geometry
Select LOD with the closest scale
Read geometry
Generalise

Figure 3.6: MRDB spatial object selection algorithm

this is illustrated in Figure 3.6.

Retrieving spatial data at a particular scale begins by querying a spatial index to determine which spatial objects are within the query's extents and matching any specified attributes. Once this is done and the appropriate LoD is identified, (for example, LoD_i) the retrieval of the geometry begins by attempting to read objects from the LoD cache for the specified scale. If the geometry for all the objects has be found at this LoD cache then the algorithm exits. Otherwise the algorithm moves to the next lowest LoD cache (LoD_{i-1}) and attempts to retrieve remaining objects there. Objects retrieved at this LoD need to be further generalised[4], cached in LoD_i and then returned. The algorithm will continue to recurse until all the objects are located and generalised to the appropriate scale.

The above approach has the advantage of simplifying the map rendering process, while having no overhead for spatial applications that only require spatial data at a single resolution. A limitation of this approach is that copies of spatial objects stored at multiple LoDs result in redundant storage, thus consuming memory. However, this redundancy is a trade-off with computation, replacing computationally complex generalisation steps with storage. The MRDB uses a metric spatial model to support map rendering. However, graph-based spatial operations such as route generation require a

[4]See Section 3.2.3 for a detailed presentation of the model-oriented generalisation mechanisms employed.

topological model. A metric model is the more generic option as topological models can be derived from the metric models but not the other way around. This approach also features weak relationships between copies of spatial objects at different LoDs, which makes it complicated to remove or edit geometry, as caches of effected objects at different LoDs will need to be invalidated. Methods for propagating updates in MRDBs have been proposed by Kilpelainen [117] and Harrie [101]. The MRBD presented in this thesis propagates updates by invalidating LoD caches containing geometry that has been modified. An investigation of the applicability of more advanced methods remains future work.

The MRDB addresses research question RQ-2, the design of algorithms to access spatial models at multiple levels of detail on hand-held devices, by providing a data store that dynamically generates and caches smaller scale representations via model generalisation. Stepped LoDs approximate continuous scale adaptation, and a spatial index effectively prunes spatial objects based on containment and attributes. The design of the spatial index is presented in the next section.

3.2.2 Spatial Index

Spatial indexes speed up access to spatial data by using minimum bounding rectangles to approximate more complex spatial objects. This results in a minimal representation on which operations, such as containment, can be easily computed.

There are two levels of spatial index used by the MRDB (Figure 3.5). The first is a flat index that performs a one-to-one re-mapping of spatial object locations for each LoD. For example, this spatial index may indicate that the object stored at position 1 in the source geometry (LoD_0) is located at position 4 in LoD_1. The second spatial index is a hierarchical, tree-based spatial index of the geometry at LoD_0. This index provides a single structure that is queried for spatial data and responds with a list of geometry records. This index supports insertions and removals as the environment model is dynamic (for example, objects move and spatial data is acquired).

A number of tree-based spatial index algorithms exist. For example B-Tree [20], R-Tree [97] or Quad-tree [74]. We include an R-Tree spatial index with the MRDB. Figure 3.7 illustrates how the data structure splits space with hierarchically nested, and possibly overlapping, minimum bounding rectangles (otherwise known as bounding

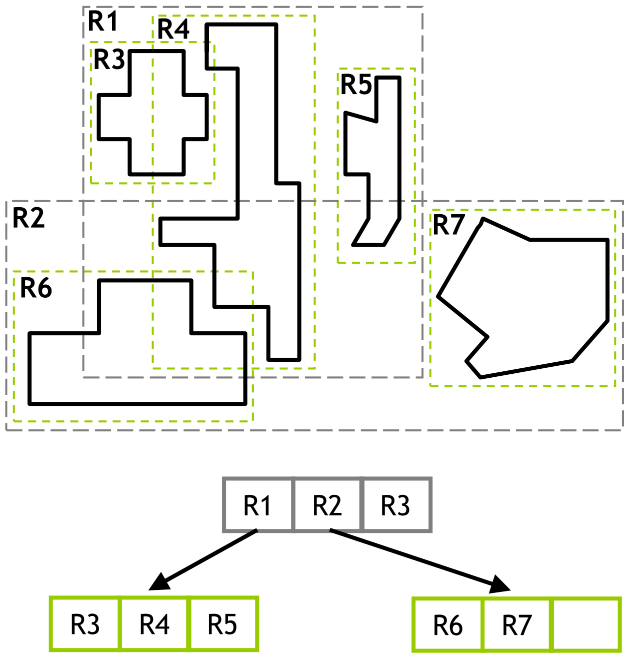


Figure 3.7: Illustration of R-Tree spatial index hierarchy for 2D geometry

boxes). The main advantage R-Tree has over other algorithms is that paged implementations are possible that minimise I/O overhead. The R-Tree is also height balanced, minimising the depth of the tree and thereby the time to query the data structure. This is important on a mobile device as data structures will be stored in flash memory that is slower to access than RAM on a desktop computer. The R-Tree algorithm was also chosen by the CRUMPET project (Section 2.1.3) to support topological queries on a PDA [207].

Alternatives to a hierarchical spatial index include a flat index file, such as the index structure incorporated into the open ESRI Shapefile standard [69]. Such an index structure is not scalable, as the entire index may need to be read and a containment test performed on every record to locate a single object. Other storage alternatives either do not take disk paging into account or perform poorly for updates and insertions.

However, the two level spatial index design adopted here has a number of limitations. As the hierarchical spatial index is not a standard part of the ESRI Shapefile format, an index must be built for each Shapefile the first time it is opened. This consumes processing resources for each unseen fragment of spatial data. On the other hand, queries are likely to be much more frequent than updates, and once created, the index persists in non-volatile memory and so survives device's reboots.

The hierarchical spatial index contributes to providing access to a local environment model on mobile devices, satisfying research question RQ-2 by facilitating retrieval and insertions using minimum bounding boxes to approximate more complex spatial objects. The two levels of spatial indexing used by the MRDB combine the strengths of each type of index.

3.2.3 Model-Oriented Generalisation

A core component of the MRDB is the ability to automatically generate new LoDs. This is achieved by model-oriented generalisation[5], which is the modification of the representation of spatial data, to derive a lighter model [53, 94, 127].

Generalisation algorithms contribute to providing spatial operations on mobile devices in two ways. Firstly, generalisation can reduce the complexity of spatial data which results in better performing spatial operations. Spatial data can be highly detailed, containing many coordinates. Manipulating this data requires complex mathematical operations to be performed on each individual coordinate. Reducing the resolution (number of individual coordinate points) of the spatial data results in reduced processing time and power consumed by spatial data manipulation operations [4]. Model-oriented generalsiation, unlike cartographic generalisation [94], can be fully automated [230].

Secondly, generalisation can derive representations of spatial objects that are suitable for viewing at different scales. For example, if highly detailed spatial data was rendered directly, without any generalisation, on a small screen, the data would need to be uniformly scaled (or shrunk) to fit on the display. This would result in a cluttered, illegible interface. The generalisation process reduces complexity by scaling the data while at the same time maintaining the representative integrity of the mapped area; exploiting a user's inability to distinguish simplifications in less important parts of the map [209].

Generalisation consists of a number of discrete operations or transformations that may be applied to spatial data [150]. The AGENT project defined a comprehensive set of generalisation operators [15]. For example: *classification*, selecting a subset of feature classes; *simplification*, a simplified representation of a spatial object using

[5]The International Cartographic Association has defined the process of generalisation as "the selection and simplified representation of detail appropriate to scale and/or the purpose of a map" [107].

less points; *collapse*, the reduction in number of dimension used to represent features; *enhancement*, modification of a feature to improve cartographic readability; *selection / elimination*, removing unimportant objects; *displacement*, moving objects to solve conflicts between objects that are too close; and *aggregation*, combining a set of objects to one object.

Generalisation is used by a number of related projects in the preparation of spatial data for use on mobile devices. Chalmers et al. [40] used generalisation to adapt maps for different bandwidths by reducing the file size. Zipf et al. stressed the need for simplification of geographic data to represent reality on a map at different scales and presented a range of algorithmic techniques for performing generalisation [245][6]. Urquhart et al. [224] takes a similar approach to developing user-centered cartographic representations for mobile devices. The work by Vacallo [230] on generalisation of spatial data (See GiMoDig, Section 2.1.6) is also related. Michela Bertolotto and Max Egenhofer [28] use similar techniques by employing geographic data generalisation to generate simplified representations of vector data. Research in the area of generalisation has not yet resulted in the ability to automatically generalise dynamically changing representations of the user's environment on hand-held mobile devices [176]. Existing scale transitions for specific classes of objects remain too complex to be executed in real-time on dynamically changing representations. Consequently, they do not meet the requirements of response time for flexible zooming and adaptation [168]. The current thesis proposes an innovative generalisation approach featuring just four lightweight generalisation operations, designed to be executed on mobile devices. Together these operations support on-demand generalisation of dynamic scenes through progressive simplification of geometry for each LoD in the MRDB.

Figure 3.8 is a flow diagram illustrating the sequence of generalisation operations employed. The following four generalisation operations were chosen to maximise the data reduction while incurring the least computational cost.

These operations are applied on-demand in a hierarchical manner to the MRDB in a process described as "Continuous Generalisation" by Sester & Brenner [208]. To illustrate this chaining of generalisation operations, we define for a spatial extent, E containing geometry for n shapes consisting of m coordinates, the maximal rep-

[6]See Kray's thesis [120] for an alternative set of generic adaption strategies for spatial information.

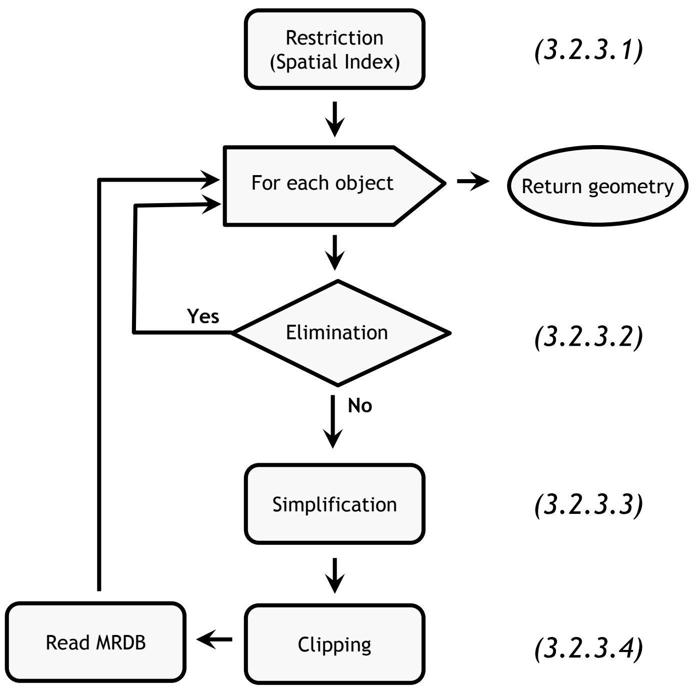


Figure 3.8: Generalisation algorithm flow chart

resentation $E^{n,m}$. We define the minimal representation, $E^{1,k}$ where k denotes the minimal number of vertices that is still cartographically sensible for a shape, with $k \leq m$. When E is generalised, we start with it $E^{n,m}$ and successively generalise the geometry until $E^{1,k}$. This results in a sequence of generalised shapes for this extent, $E^{n,m} \rightarrow E^{n-1,m-1} \rightarrow \ldots \rightarrow E^{1,k}$. At each level of generalisation, the set of geometry represents a distinct LoD. This processing chain allows the quick derivation of a shape at any desired generalisation by selecting an appropriate level in the multi-scale data structure and further generalising to the necessary scale.

The following sections explain the four generalisation operations:

1. Restriction

2. Elimination

3. Simplification

4. Clipping

3.2.3.1 Restriction

Restriction has been proposed by Chalmers et al. as a method of adapting map-based interfaces on mobile devices by selecting a number of primitives that can be removed

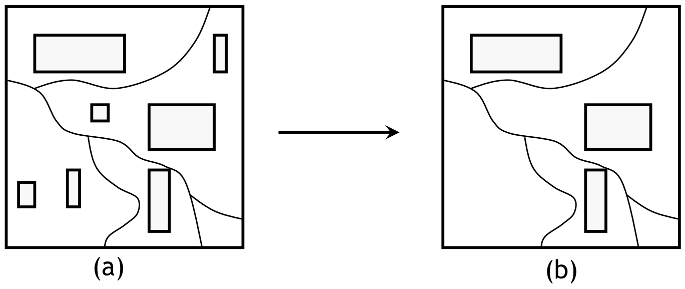


Figure 3.9: Elimination generalisation step

from the interface based on the user's context [40]. This generalisation step takes advantage of the semantics of the spatial data format to perform a selection of data based on its attributes. For example, consider a map view showing the whole world. It would not be appropriate to include streets on such a map as the volume of information would overwhelm the user's ability to read the map. However, if the scale is changed and we zoom in to a small area, the street data would now be useful. The process of omitting or restricting the data that is returned based on visibility at particular scales is the simplest generalisation step that the generalisation algorithm takes.

3.2.3.2 Elimination

Elimination is a similar generalisation step to restriction as it eliminates features that are too small, too short, or too insignificant to be presented in the final map [71]. The definition of how small a spatial object needs to be to be considered insignificant is application specific. In contrast with restriction, elimination does not quickly reject spatial data based on an attribute, but performs a filtering of the geometry based on the screen area it will occupy when rendered. This results in small interface artifacts, such as, small buildings and short roads being removed from the interface (that is, objects too small to be seen). Figure 3.9 illustrates this process. On the left, (a), a detailed map is shown. On the right, (b), the results of elimination are shown. The maximum size of eliminated objects is adjusted through the modification of a configuration parameter. This parameter is used by a generic elimination filter as an area threshold. Objects with a rendered, on-screen area below this threshold are removed. For example, a threshold value of $1px^2$ would eliminate all spatial objects that, when rendered at the current scale, would occupy less than $1px$ on the devices screen.

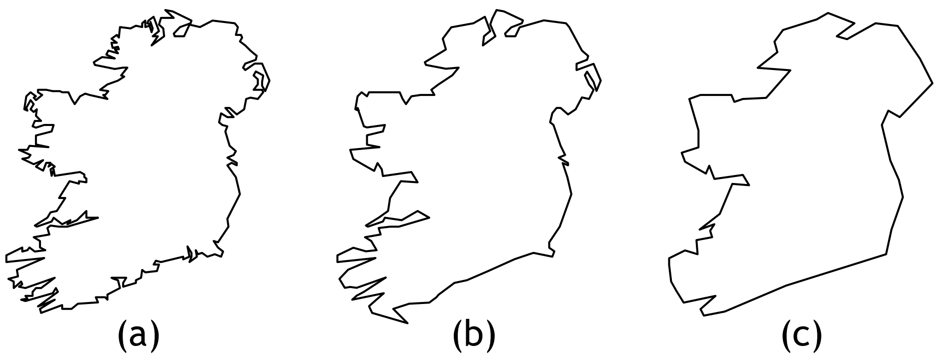


Figure 3.10: Simplification generalisation step

3.2.3.3 Simplification

Simplification is a generalisation step where the complexity of individual shapes is reduced. Simplification not only produces information suitable for display at different levels of detail but reduces the complexity of the information. This is of significant advantage on mobile devices, where replacing a large model with a smaller model might have little visual impact but be significantly faster to render.

Figure 3.10 illustrates the effect of simplification on a shape (Generated with MapShaper [30, 102] using Douglas-Peucker line simplification [62]). The figure on the left, (a), is the original shape. The figure in the middle, (b), shows the shape after simplification and (c) shows the same shape after further simplification. As the shape is simplified the number of points is reduced resulting in the loss of fine detail. Despite this, the shape remains recognisable, an important feature of simplification algorithms.

The simplification algorithm uses computational geometry methods to reduce detail in a 2D space. These methods are designed to eliminate points from lines and polygons while minimising the visible change in shape [181]. A commonly used method is the Douglas–Peucker algorithm [62], in which a baseline is constructed from an object's two endpoints. Points having a perpendicular distance from the baseline that exceeds some threshold are added, forming a new baseline. The process is repeated until all points fall within the threshold distance to the baseline. Other simplification algorithms use different selection criteria, such as, angular change [141] or area displacement [228] between various intermediate lines in an object's geometry.

This thesis proposes the use of two complementary simplification algorithms. The first is the *nth* point algorithm which simply removes every *nth* point from the shape. This lightweight algorithm causes distortion and so is only suitable for background

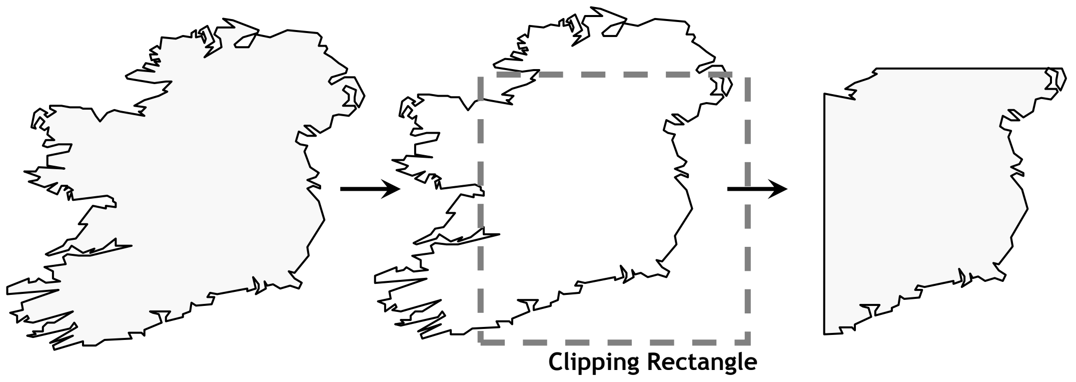


Figure 3.11: Clipping generalisation step

objects, objects with a large number of points or objects to be displayed at a scale where inaccuracies will not be seen. For geometry where the number of points or scale requires minimal distortion, the Douglas-Peucker algorithm is chosen because of its use of a tolerance factor which can be varied according to the amount of simplification required [62].

3.2.3.4 Clipping

When individual spatial objects extend beyond the viewport of the device, resources are wasted in computing projections and rendering coordinates that will not be seen. Clipping modifies the geometry so that the parts of the shape that lie outside the view are removed. Figure 3.11 illustrates a shape that extends beyond the device's viewport being clipped. The clipping of polygons and polylines is carried out using one of the many algorithms proposed to solve these problems. For example: *Liang-Barsky* [17], a parametric 2D line clipping algorithm optimised for an upright rectangular clip window; *Cohen-Sutherland* [217], coordinate based line clipping with trivial accept / reject testing to improve performance in the case where lines are totally inside or outside a clip area; *Nicholl-Lee-Nicholl* [160], a variation on *Cohen-Sutherland* that reduces the number of clip rectangle sides that need to be tested; *Weiler-Atherton* [236], an algorithm that can clip a polygon against a non-rectangular window; *Sutherland-Hodgman* [29], an algorithm for clipping polygons against a convex clipping window that is efficient when the polygon falls completely inside or outside the clipping boundaries; and *Malliot* [133], an extension to *Sutherland-Cohen* 2D line clipping.

To enable the substitution of implementations of these algorithms, a generic clipping component interface is created. All of these algorithms may then be implemented by classes realising the component interface. Although all of these algorithms can be implemented, the *Cohen-Sutherland* algorithm is chosen for line clipping because of its trivial acceptance and rejection of lines that are outside of the clipping area. The *Sutherland-Hodgman* algorithm is used to clip polygons because of its suitability for clipping geometry that does not have holes or self-intersect (two of the restrictions of our geometry model in Section 3.3.3). These two algorithms are chosen because their implementations demonstrated the best performance on a test spatial data set (Table 5.1 on page 145). However, they are limited to the simple closed polygons supported by the geometry model. The extensibility of the interface supports the substitution of other clipping algorithms that may be required for one of a number of reasons. For example, the use of an extended geometry model, a spatial data set whose characteristics favors a different algorithm or to provide an implementation that is more efficient on a specific mobile platform.

3.2.3.5 Summary

In summary, model-oriented generalisation contributes to providing access to a spatial model at multiple levels of detail (RQ-2), by making it possible to derive less complex models of spatial data on demand. These models form the LoDs in the MRDB. The main limitation of this approach is that the generalisation algorithms take into account only the geometric vector properties of spatial objects and do not guarantee the topological correctness of generalised data. This leads to the possibility of topological inconsistency between LoDs resulting in spatial data that is less suitable for rendering maps [113]. Cartographic generalisation, as opposed to model-oriented generalisation, does not suffer from these limitations, but is significantly more complex and cannot be fully automated [235, 230]. This thesis trades the accuracy of the LoDs for a reduction in computation complexity by using the simpler model-oriented generalisation process within the MRDB.

	Guide [43]	comMotion [136]	GeoNotes [68]	TellMaris [120]	Riot! 1831 [192]	CYSMN [26]
Containment			●		●	
Proximity		●	●	●	●	
Map Rendering	●	●		●		●
Route Generation	●			●		
Spatial Selection		●				
Geocoding		●				

Table 3.1: Spatially-aware mobile applications

3.3 Spatial Middleware Services

The previous section has described algorithms for manipulating dynamic spatial data at multiple resolutions on mobile devices. This section describes the application-level spatial services designed for mobile spatially-aware applications derived from these algorithms (RQ-3). We begin by analysing the requirements for middleware-based spatial services in spatially-aware mobile applications.

3.3.1 Analysis & Requirements

This section draws on the review of related work to establish a set of requirements that are used to create a middleware of spatial services for supporting spatially-aware mobile applications. The primary objective of this analysis is to identify the core architectural features required of a middleware to support the next generation of spatially-aware mobile applications.

The spatial services commonly used by spatially-aware applications are summarised in Table 3.1. These services include: navigation [43, 19, 120], communication [43, 26], mapping [43, 19, 68, 26], geocoding [136] and location specific information services [120, 26, 192, 136]. The model for mobile spatial middleware presented in this thesis incorporates support for the spatial services: adaptable map rendering, spatial reasoning, coordinate transformation, route generation and visibility determination. This set of services is chosen because: a) these are the most common services required by spatial applications, as illustrated by the review of related work (Chapter 2); and b) they provide a core set of features that are used as a foundation for the development of additional spatial services. An exception is the visibility determination service, which is not a core service, but a novel spatial service designed to demonstrate the ability of the spatial middleware to support novel interfaces and interaction metaphors. The design of these spatial services is presented in the following sections.

3.3.2 Adaptable Map Rendering

Map rendering is the process of turning vector-based spatial data into a raster map for display on screen. Map-based interfaces are common to a range of spatially-aware mobile applications. Unlike their paper map counterparts, map-based mobile interfaces have the advantage of being able to adapt themselves to the user's preferences and environment. For example, map-based mobile applications commonly centre the viewport on the user's location. In addition, it may be desirable to automatically zoom, scale or rotate a map interface for moving users. To achieve this, spatially-aware mobile applications require maps ranging from simple black and white sketches [129] to 2D-maps [89] to animated 3D-maps [120].

Reichenbacher published a number of seminal articles on the topics of cartographic theory, geographic information communication in mobile environments and adaptive methods for cartographic visualisation [189, 191]. As part of this work the different ways in which map-based interfaces can be adapted were defined:

- Selecting map features.
- Adjusting the level of detail.
- Classifying and grouping information.
- Highlighting or prioritising map features.
- Modifying map extent.
- Adjusting map scale.

Providing for these user services on a mobile device is challenging, as user mobility requires that adaptions be performed "on-the-fly". A layered architecture, inspired by the GeoTools [18] open source library of methods for manipulating geospatial data, is

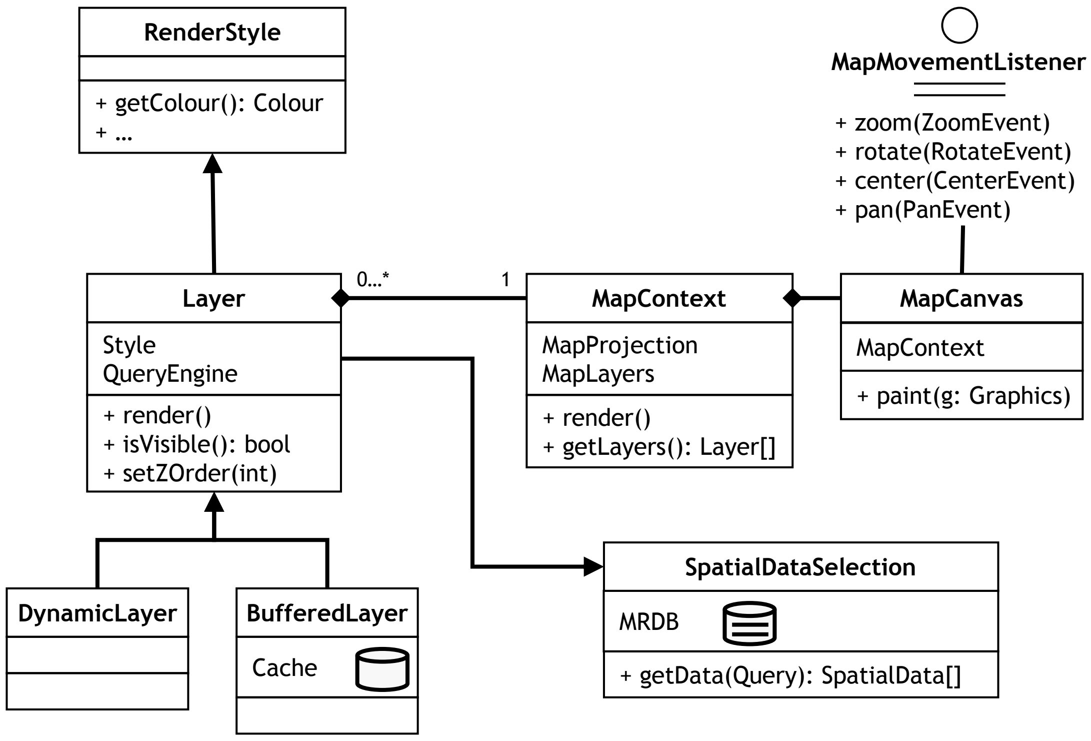


Figure 3.12: Adaptable map rendering layers

adopted. The layered architecture deviates from the GeoTools architecture by incorporating the concept of rotatable map layers to support track-up interfaces on hand-held devices.

Figure 3.12 illustrates the main entities involved in the layered architecture for adaptable map rendering. The key metaphor at the centre of this architecture design is that of a layer. A layer represents a transparent overlay of information that has a unique set of geometry, a definition of how that geometry should be rendered and a `MapContext`. The adaptable nature of map rendering is facilitated by the introduction of two types of layer supporting different rendering policies: dynamic and buffered. Dynamic layers render information that changes quickly (for example, user's location) and are always re-drawn in response to changes in the map projection or user interaction with the `MapCanvas`. Dynamic layers can also request that they be redrawn independently of other layers. Buffered layers are intended to display background, slowly changing information. The buffered layer rendering policy is to store a cache rendered output and to reuse this cache whenever possible. For example, changes in dynamic information do not trigger a redraw of buffered layers. The cached image of the buffered layer is used as a background to redraw the dynamic information and refresh the interface. This layered design for adaptable map rendering presented in the current thesis enforces two different rendering policies, minimising the computation resources consumed by drawing static spatial objects (RQ-3). Reducing the volume of geometry to be redrawn contributes to a more efficient use of limited processing resources and a more responsive application interface.

3.3.3 Spatial Reasoning

Spatial reasoning goes beyond the traditional use of spatial data for map viewing in order to include its use as an analytical tool for characterising spatial relationships [27, 194, 24]. It has been suggested by Greaves & Stopher [92] (as quoted by Wang & Cheng [231]) that GIS can be used as a "spatial decision support tool" for activity-based applications[7].

Spatial reasoning is a common feature in spatially-aware applications. Support for operations such as, overlap, intersection, containment and distance has been a feature of projects including Deep Map [134], CRUMPET [178], GeoNotes[68], Nexus [161], GiMoDig [208] and M-Spaces [200]. The approach that these projects take is to have a geometric model of the environment on which these operations are applied. Guide [43] makes use of a geometric model to support route guidance and provides information about specific physical locations. EasyLiving [35] from Microsoft Research models its environment using a geometric model, in which all entities are modeled as extents. MiddleWhere [185] also uses a geometric model to support spatial reasoning. The ability to determine geometric, topological and logical relationships between physical objects is invaluable to spatially-aware applications and yet poses a challenge to design in a lightweight manner without severely restricting the range operations.

This research reported in the current thesis addresses this challenge using a geometric environment model that supports operations for determining spatial relationships between objects. The geometric model is based on the types of geometry supported by the MRDB (for example, lines, polygons and points). Once retrieved from the MRDB, these spatial objects facilitate the use of geometric operations to reason over the user's environment, determine higher level contexts and triggering behaviour.

[7]Activity-based applications are based on a user's activities and are generally 'tied' to his or her personal computer [16]. Activity-based applications subsume and include mobile, context-aware applications of which spatially-aware applications are a class.

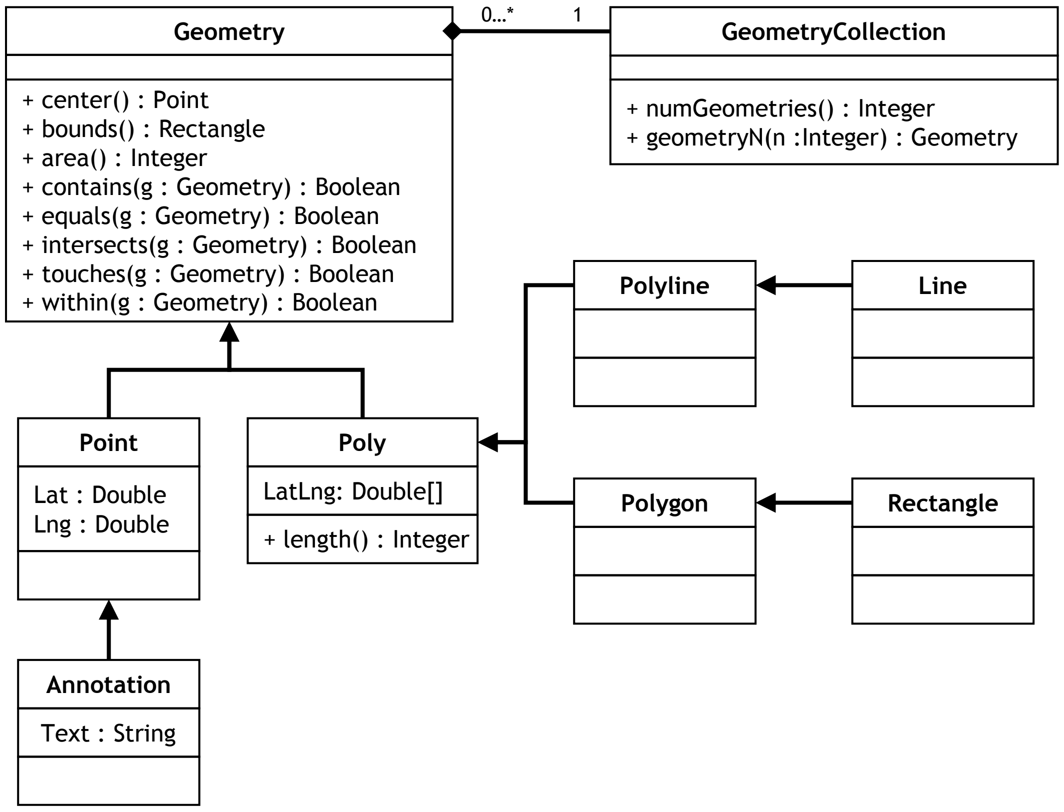

Geometry
+ center() : Point
+ bounds() : Rectangle
+ area() : Integer
+ contains(g : Geometry) : Boolean
+ equals(g : Geometry) : Boolean
+ intersects(g : Geometry) : Boolean
+ touches(g : Geometry) : Boolean
+ within(g : Geometry) : Boolean
0...*
1
GeometryCollection
+ numGeometries() : Integer
+ geometryN(n :Integer) : Geometry
Polyline
Line
Point
Lat : Double
Lng : Double
Poly
LatLng: Double[]
+ length() : Integer
Polygon
Rectangle
Annotation
Text : String

Figure 3.13: Geometry model showing spatial relationship functions

Figure 3.13 illustrates the geometric model. The types of geometry supported include Line, Polyline, Rectangle, Polygon, Point and Annotation. Each of these objects inherit support for the functions listed as members of the Geometry object. This particular set of objects was chosen as there is a one-to-one mapping between these objects and the geometry types supported by the MRDB, which in turn are inherited from the underlying Shapefile specification (Section 4.1.1). These object types are also closely related to the OpenGIS Simple Features specification [171]. The Simple Features specification defines a common feature model containing vector data elements, such as points, lines and polygons, as well as a set of methods for testing spatial relations between geometric objects. These methods allow the spatial relationship between pairs of features to be characterised. This set of methods, defined by the Simple Features standard and incorporated into our geometric model are:

- *equals (g: Geometry)*
- *disjoint (g: Geometry)*
- *intersects (g: Geometry)*

- *touches (g: Geometry)*
- *within (g: Geometry)*
- *contains (g: Geometry)*

In addition to these geometric operations, a helper proximity component calculates the distance between points. The environment model uses polar coordinates (that is, latitude and longitude). Calculating the distance between pairs of coordinates requires calculating the shortest route between two points on the surface of the sphere (the Earth).

A spherical distance algorithm is used to calculate this distance based on the spherical law of cosines[8]. The algorithm assumes a spherical earth, ignoring ellipsoidal effects. This method of calculating spherical distance is not commonly used as it suffers rounding errors where the distance is small. An equation, known as the haversine formula, is preferred but is more computationally complex in the case where it is implemented in software without access to tables for the haversine function [214]. The simpler spherical law of cosines formula gives accurate enough results (down to distances as small as 1 meter when computed using floating-point numbers) and so is more appropriate for use on hand-held devices [226].

There a number of trade-offs and limitations associated with the geometric model chosen. These limitations are largely associated with the restricted types of geometry supported. Figure 3.14 illustrates some of the geometry types supported and not supported. Curves and complex polygons are excluded from the model but non-simple polylines and polygons with multiple interior boundaries are allowed. The restriction of geometry types in the model reduces the size of the data structure and allows for a simpler implementation of geometric operations.

Spatial reasoning provides the basic operations that enable the design of common application-level spatial services using the geometric model of the user's environment (RQ-3).

[8]The great circle distance, D is calculated as $D = \arccos(\sin(lat_a)\sin(lat_b) + \cos(lat_a)\cos(lat_b)\cos(lon_b - lon_a))$. The real world distance estimate can then be calculated using, $D(2\Pi r/360)$ where r is an estimate of the earth's radius.

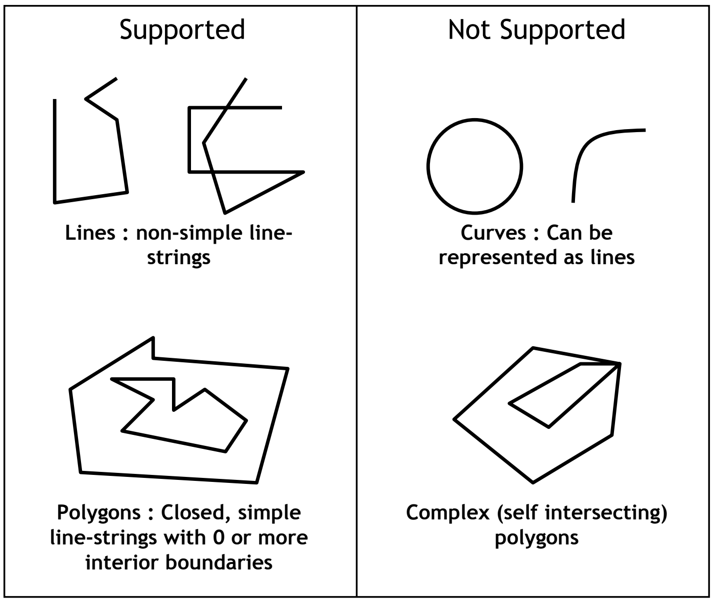

Supported
Not Supported
Lines : non-simple line-strings
Curves : Can be represented as lines
Polygons : Closed, simple line-strings with 0 or more interior boundaries
Complex (self intersecting) polygons

Figure 3.14: Supported geometry types

3.3.4 Coordinate Transformation

Coordinate transformation refers to the conversion of points on the earth's surface between different geodetic coordinate reference systems and a mobile device small screen coordinate system. To minimise the need for these operations, the model for spatial middleware presented in this thesis standardises all coordinates in WGS84 (World Geodetic System 1984), the geographic coordinate system used by GPS. This eliminates the need to translate between spatial data in multiple coordinate systems and supports easier integration with GPS (the most common positioning system used by mobile applications [129, 68, 26, 19, 120]).

The remaining need for coordinate translation is in the mapping from longitude and latitude coordinates to the hand-held device screen coordinate system. The method of flattening coordinates on the surface of a round body (that is, the Earth) on to a 2D plane is known as map projection. Several hundred map projection algorithms have been published, many of which may be infinitely varied by choosing different points on the Earth as the centre or as a starting point. All projections lead to varying degrees of distortion of one form or another [215].

However, spatially-aware mobile applications have unique requirements that traditional map projections do not take into account. In particular, applications are highly interactive and do not always feature north-up interfaces. It must be possible to ro-

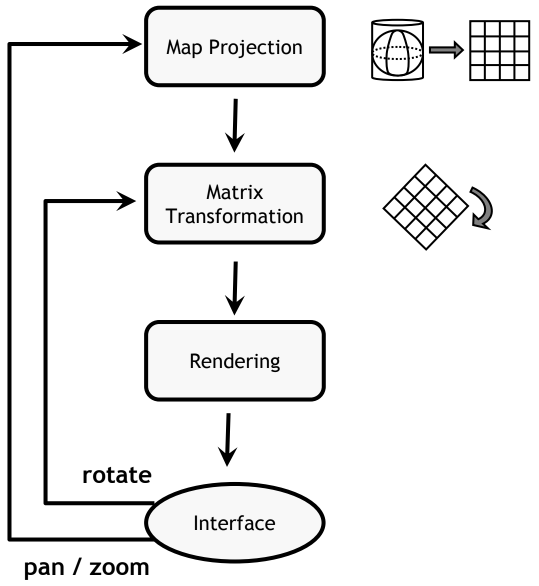


Figure 3.15: Two-step coordinate transformation algorithm

tate screen coordinates about an arbitrary point and to easily compute the inverse of any transformation. The inverse of a coordinate transformation relates screen coordinates back to real world locations and is required to handle user interaction with the interface, which will be in screen coordinates, to the spatial objects that are being manipulated, represented in real-world coordinates.

A two-step coordinate transformation algorithm has been designed to satisfy the requirements of a generic middleware for spatially-aware mobile applications. Figure 3.15 illustrates this algorithm.

The first step involves applying a traditional map projection algorithm to flatten spherical coordinates (latitude and longitude) onto a 2D plane. Two map projection algorithms are proposed: equi-rectangular, and mercator [186]. The equi-rectangular[9] projection is a very simple map projection that converts the globe into a cartesian grid of perfect squares (that is, meridians become equally spaced vertical straight lines, and parallels become equally spaced horizontal straight lines). This projection features simple calculations that preserve energy and computational resources of mobile devices. Although area and shape are distorted (increasing towards the poles) it is sufficient for display of small areas. This trade-off between accuracy and speed is based on the intuition that pedestrian-scale map extents are more common than global-scale maps in spatially-aware applications on hand-held devices. The mercator projection

[9] Also referred to as the equidistant cylindrical projection [186].

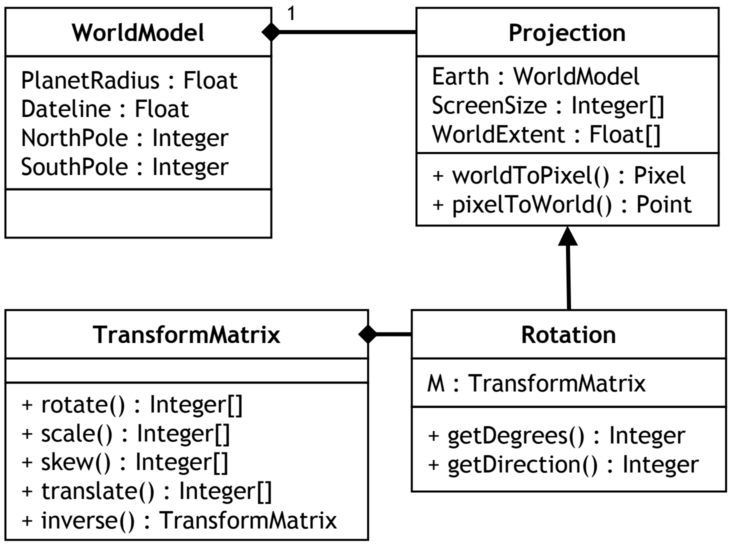


Figure 3.16: Coordinate transformation design detail

is a conformal projection that preserves angles but distorts the size of geographical objects (particularly in areas closer to the poles). The mercator projection is used by Google [91], Yahoo [241], and MapQuest [135] web applications. Both these map projection algorithms are implemented as pluggable components and an evaluation of their comparative performance is presented in Section 5.1.3.4.

The second step of the coordinate transformation algorithm is matrix transformation where a transformation matrix is used to shift, magnify, or rotate the coordinates, or perform several of these changes at once. These transformations are cumulative and can easily be reversed. As the map projected coordinates from step one are cached, it is possible to support highly dynamic interfaces that rotate, pan and zoom without having to recompute the map projection.

This two step approach supports the dynamic nature of spatially-aware mobile applications by allowing coordinates to be quickly transformed in support of map rendering. In particular, changes in map orientation are accommodated by re-computing only the second step of the coordinate translation without re-computing the map projection step. Coordinate translation accuracy is traded for simple calculations that are performed more quickly. Coordinate translation is a core service included in the model for spatial middleware that is reused by application-level services to support map-based interfaces (RQ-3). As such, its design focuses on performance, choosing the least computationally expensive map projections and supporting rotation through a lightweight matrix translation as a second step.

3.3.5 Route Generation

Mobile spatially-aware applications increasingly incorporate navigation support (GUIDE [43], REAL [19], TellMaris [120]). As our model for spatial middleware aims to support this class of application, it must incorporate a route generation service that can produce sets of connected, navigable way-points between locations. As a service of the middleware, route generation can only use the multi-scale geometric environment model of the MRDB (Section 3.2.1) and the spatial reasoning functions (Section 3.3.3).

The most common strategies for performing route generation use graph traversal algorithms. These algorithms require a graph-based representation of the environment where roads or paths are represented by edges and the junctions by nodes. Graph-based route generation is a special case of the shortest path problem in which a shortest path is required between two nodes in a directed graph. Problems in this area have been extensively studied by computer scientists resulting in a set of algorithms that can be applied to solve this problem including: Floyd's algorithm, Dijkstra's algorithm and $A*$ search. Where a route other than the shortest path is required the cost function applied to each edge in the graph can be modified.

This thesis uses Dijkstra's algorithm [60] with a configurable cost function to search for routes in a topological (that is, graph based) spatial model. However, the MRDB environmental model is geometric, and so does not contain any concept of connectedness. Route generation using our environment model is achieved by initially discovering topological relationships from line intersections in the underlying spatial data. A topological model building algorithm retrieves a list of polylines representing paths and builds a graph of navigable routes. This allows an innovative route generation service to be developed that would normally require a much more sophisticated topological model to use the simple geometric model of the MRDB.

Figure 3.17 illustrates the clustering algorithm used to build the topological model from the MRDB environment model. The algorithm first selects all the paths stored in the model and then attempts to connect them up by identifying locations where paths touch, intersect or overlap. Coordinates where these artifacts are found become the nodes of the graph. Once created the graph is pruned by removing nodes that do not indicate a junction (that is, nodes of degree two).

This graph can then be searched using the `DijkstraPathFinder` (Figure 3.18),

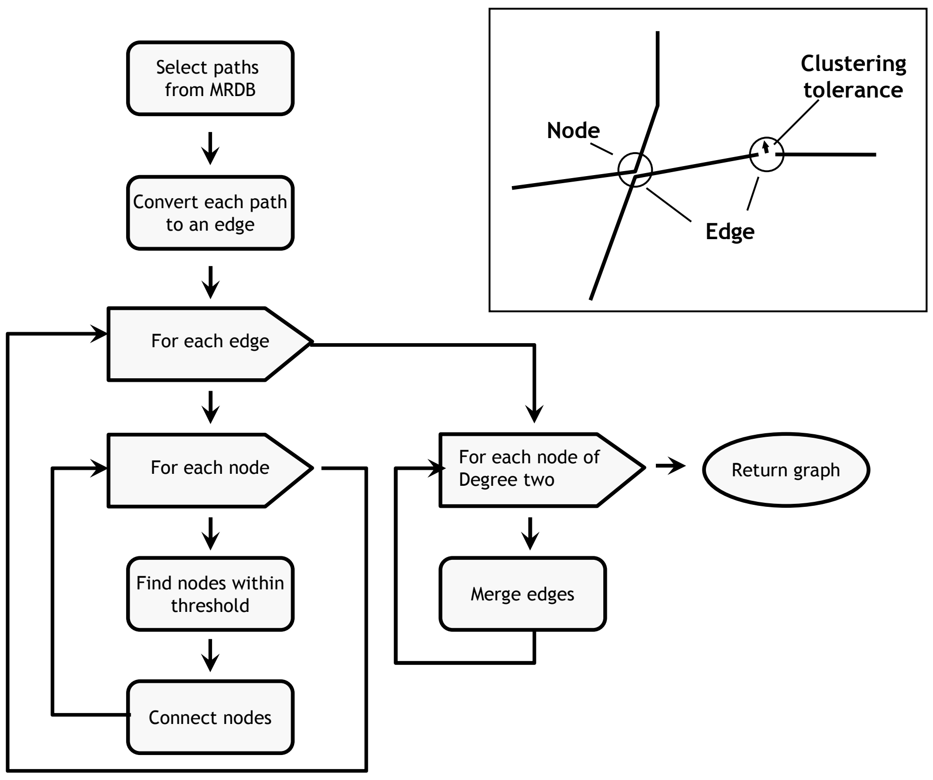


Figure 3.17: Topological model building

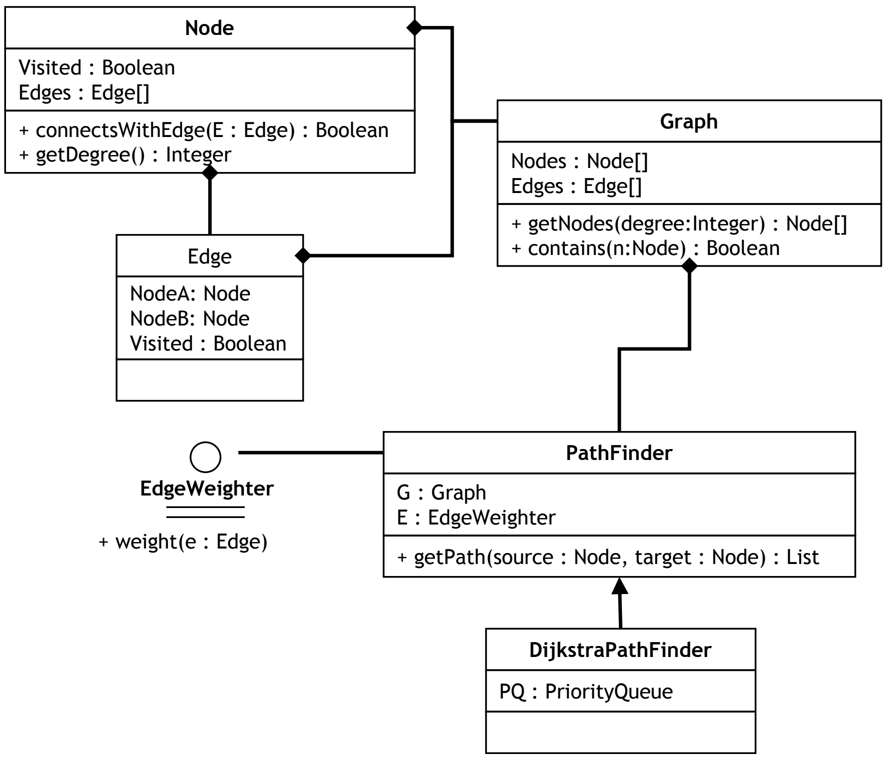


Figure 3.18: Graph-based route generation

which finds a path between two nodes in a graph that minimises a single parameter. An `EdgeWeighter` interface allows the route generation process to take into account parameters other than just distance.

The limitation of this approach is in the computation overhead in constructing a topological model before route generation can be performed. However, this process only needs to happen once, as the topological model can be persisted in memory, and can be carried out while the device is idle, in advance of a route generation query. Once complete, the topological model is integrated with the model for spatial middleware to provide navigation services to mobile spatially-aware applications. The dynamic inference of topological relationships in the geometric MRDB environment model enables navigation services to be provided to mobile spatially-aware applications using the set of algorithms for spatial operations included in the model for spatial middleware (RQ-3).

3.3.6 Visibility Determination

The current section describes the design of an algorithm to support an innovative spatial visibility service that determines the visibility of real world objects from the users point of view (PoV) using 2D environment model presented in Section 3.1. Although visibility has been used in a small number of mobile applications [85, 23], our middleware approach is innovative in that it provides a generic service to spatially-aware applications without relying on a server-based environment model. In addition, these prototypes focused on usability and user interface research issues as opposed to the efficiency and accuracy of the service's design and implementation. Knowledge of the visibility of geographic features from a certain point of view has been shown to be crucial for enabling spatially aware applications [84]. A naïve algorithm for determining the visibility of an object in a 2D environment model may draw a line from every point in an object and test to see if that line intersects any geometry (other than the object being tested). However, this solution is not suited to being performed in software on hand-held mobile devices. More efficient methods for determining the visibility of objects can be built using techniques developed in the computer graphics field, where visibility is used to bring the cost of rendering a large scene down to the complexity of the visible portion of the scene. Before exploring considered approaches and related work, some of the relevant terminology is introduced[10].

Visibility is the existence of a line of sight from a point (i.e., the user) to an object. The visible set is defined as the set that includes all of the visible objects within a specified field of view, from a specific point of view. Conservative visibility is the set that includes at least all of the visible set plus some additional invisible objects (often referred to as the potentially visible set (PVS) [5]). An occluded (visually obstructed) object may be classified as visible, but a visible object may never be classified as occluded. The PVS of a scene is determined by culling invisible geometry. Visibility culling is the term given to the process of quickly rejecting invisible geometry and is used in computer graphics as a pre-processing step when performing hidden-surface removal (HSR) as part of the scene rendering process [78]. HSR algorithms identify the exact portions of visible polygons and are computationally complex, whereas, visibility culling merely identifies polygons that are definitely not visible. There are two classical strategies to performing visibility culling: view-frustum and back-face culling [78]. Figure 1.6, in Section 1.3.4 (Chapter 1), illustrates the difference between these strategies. View-frustum culling eliminates objects outside of a defined field of view. Back-face culling eliminates objects that are not visible because they are behind another object.

A spatially-aware application's environment model stores knowledge about the user's world – its structure and its geometry. Visibility information augments this model with knowledge about the environment's visual appearance and the user's relative position [84]. Incorporating visibility information benefits spatially-aware mobile applications in a number of ways. Firstly, information can be tailored considerably better to the user's context, since points of interest that are visible (for example, places in the same street as the user) are most likely more relevant than those that are hidden.

Secondly, visibility information enables new types of user interfaces. For example, the local visibility model used in the P2D project supported spatial query operations based on visibility and field of view [212, 84]. This model enabled the development of real world point and click information retrieval applications but relied on server-based environment model. Visibility information can be exploited when adapting existing map-based interfaces by changing the interface's scale to fit the currently visible set on the display of the device, or by rendering objects at different resolutions depending on

[10]Also see Appendix C, Glossary.

their current visibility [191]. Maierhofer et al. supports the use of visibility information to add value to interactive egocentric maps [132].

Thirdly, visibility information contributes to the usefulness of navigation instructions by making it possible to determine which local landmarks are currently visible [22]. The intuition is that visible landmarks will help users orientate themselves. In addition, it may be more intuitive and less frustrating to reference visible objects in navigation instructions. It has also been argued that visibility information can lead to less cluttered maps and an enhanced user experience, through the removal of irrelevant information [23]. Finally, visibility information has the potential to be used as a filter to tailor information based on relevance (where information relating to visible objects is more relevant) [87].

Visibility determination is a difficult problem to solve on resource limited handheld devices, as small changes in the user's location, movement of dynamic objects in the model and the orientation of the device can cause large changes in the visible set. The current thesis contributes a visibility determination algorithm for spatially-aware applications, based on the use of a depth buffer, on which both forms of visibility culling are performed in order to reduce the set of geometry to a conservative possibly visible set (PVS). The algorithm is confined to 2D spatial data and uses axis-aligned, regularly shaped bounding volumes, instead of complex spatial objects to improve performance.

The visibility service includes a combination of algorithms and data structures to determine the visibility of spatial objects on-demand in a dynamic scene. These algorithms and data structures are:

1. *View Frustum Culling*: The removing of objects that are not within the user's field of view.

2. *Depth Buffer*: The ordering of objects based on their distance from the PoV.

3. *Occlusion Culling*: The use of back-face culling or ray casting to further remove objects that are occluded by closer objects.

The result of applying this combination of algorithms is a lightweight visibility determination middleware service for spatially-aware applications designed using only the

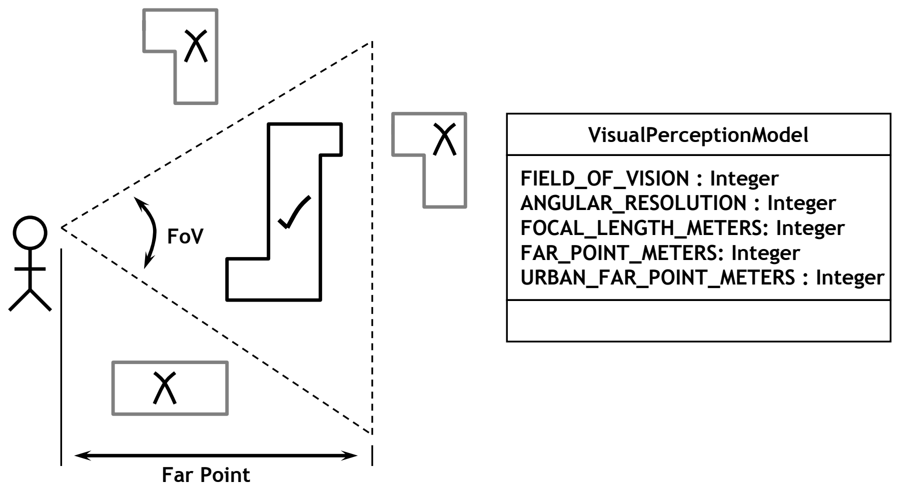

FoV
Far Point
VisualPerceptionModel
FIELD_OF_VISION : Integer
ANGULAR_RESOLUTION : Integer
FOCAL_LENGTH_METERS: Integer
FAR_POINT_METERS: Integer
URBAN_FAR_POINT_METERS : Integer

Figure 3.19: View frustum culling perception model design

algorithms, spatial operations and environment model included in the model for spatial middleware.

3.3.6.1 View Frustum Culling

The process of visibility determination begins with view frustum culling. Figure 3.19 illustrates the effect of eliminating objects outside a field-of-view (FoV), or beyond a defined distance from the user. A model of human visual perception (Figure 3.19, right) is used to determine the FoV and a reasonable cut off point beyond which objects should not be considered visible. The visual perception model contains a number of attributes that are used to compute the FoV that is used to filter invisible objects:

`FIELD OF VIEW` The field of view is defined as the angular extent of the observable world that is seen at any given moment. The model uses a value of 140 degrees[11], which corresponds to the angle of view of the human eye [196].

`ANGULAR RESOLUTION` The minimum distance between distinguishable objects (Often referred to as visual acuity). In humans, the maximum acuity is 30–60 cm at a 1 km distance [238].

`FAR POINT` This is the furthest distance at which an object is seen (should be infinity). In this case we assume it is the horizon, estimated as the distance at which a two story building (12 meters high) will disappear over the horizon [112].

[11]140 degrees is considered perfect vision. 40 degrees of the 140 would be considered peripheral vision.

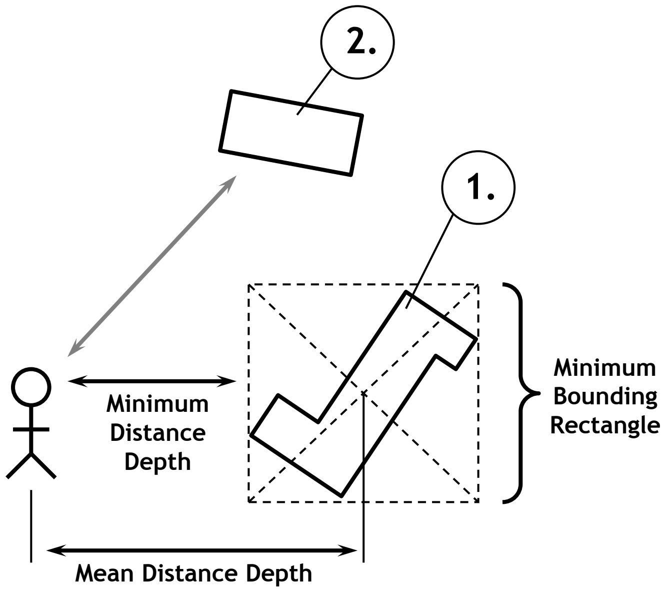


Figure 3.20: Depth buffer distance comparisons

`URBAN FAR POINT` In urban environments the maximum distance a person can see is not related to the distance to the horizon, but is defined by the surrounding buildings. A reasonable estimate of a usable figure can be made by analysing the spatial data for building density and straight lines.

3.3.6.2 Depth Buffer

A depth buffer, inspired by the zBuffer data structure used in computer graphics [39], is a sorted list of geometry objects ordered by their distance from a point of view. In computer graphics, z-buffering is often per-pixel, whereas our depth buffer operates on a per-object basis (that is, each object has a single depth value associated with it). Figure 3.20 illustrates two possible methods that the depth buffer can use to calculate the distance to each object, the minimum distance or mean distance. In the case of using the minimum distance, four distance calculations are required. When using the mean distance, the centre of the bounding volume is used, thus reducing the number of distance calculations per object to one. Minimising the number of distance calculations reduces the computational overhead of the depth buffer as each distance calculation requires the calculation of a spherical distance between two polar coordinate angles indicating a point on the surface of the earth (See Section 3.3.3). However, the centre of an objects' bounding volume may be a poor estimate of the centre of the object itself, reducing the accuracy of the depth buffer. This inaccuracy could potentially result in objects being ordered incorrectly in the depth buffer, which in turn impacts on the accuracy of visibility culling algorithms that use the data structure. The depth buffer uses the mean distance to order objects because of its lower computational overhead. The minimal effect on overall accuracy is demonstrated in the visibility service's evaluation (Section 5.1.3.5).

3.3.6.3 Occlusion Culling

Occlusion culling is the process of filtering the set of possible visible objects ordered according to depth to remove objects that are occluded by other objects. Listing 3.1 illustrates the general approach to occlusion culling as pseudocode. In this listing, G, contains all the objects in the scene and OR is the occluded representation.

Listing 3.1: Pseudocode for a general occlusion culling algorithm.

```
OcclusionCullingAlgorithm (G)
OR=empty
for each object g in G
   if (isOccluded(g, OR))
      Skip(g)
   else
      Render(g)
      Update(OR, g)
   end
end
```

The interesting part of this algorithm is the implementation of the `isOccluded()` function. We design two possible algorithms for this function; back-face culling and ray casting, with a view to selecting the best performing implementation.

The *back-face culling* algorithm is illustrated in Figure 3.21. This process uses the depth buffer as an occluder hierarchy. Starting with the closest object (that is, largest occluder) the occlusion shadow cast by each object is computed. The process of computing the occlusion shadow for an object is illustrated on the right. The angle created between the PoV and extreme edges of the object is found by projecting sight lines to each point in the object. This angle defines the size of the occlusion shadow.

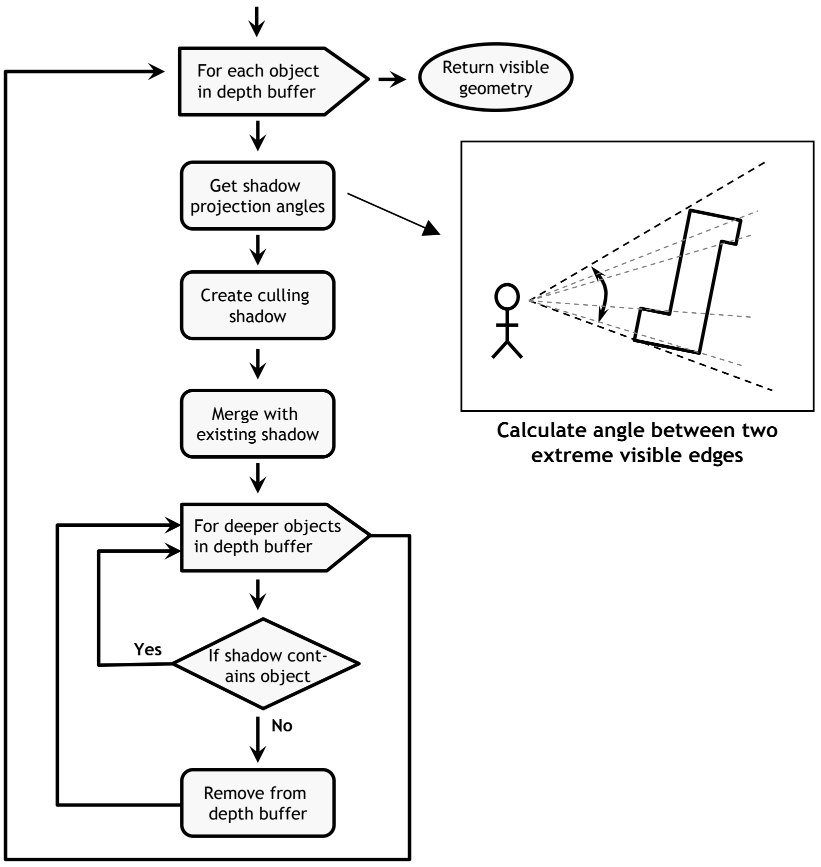

For each object in depth buffer
Return visible geometry
Get shadow projection angles
Create culling shadow
Merge with existing shadow
For deeper objects in depth buffer
If shadow contains object
Yes
No
Remove from depth buffer
Calculate angle between two extreme visible edges

Figure 3.21: Back face occlusion culling

Objects falling completely within this shadow are then removed. Each time a new occlusion shadow is created it is merged with the previous shadow using a convex hull algorithm, preventing objects on the border of more than one occluder being marked as visible. As this process continues, the depth buffer becomes smaller until only visible objects are left. This method quickly discards many objects, so is efficient in the case where there are a few large occluders. For small numbers of objects it may be quicker to just cast rays to each object and look for intersections with other geometry.

The *ray casting* approach is the second algorithm for computing occlusion. Ray

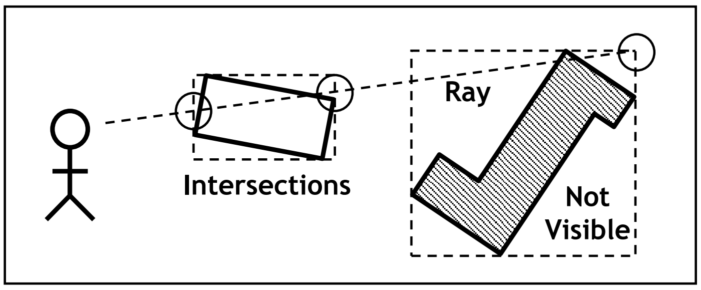


Figure 3.22: Ray casting from a single object

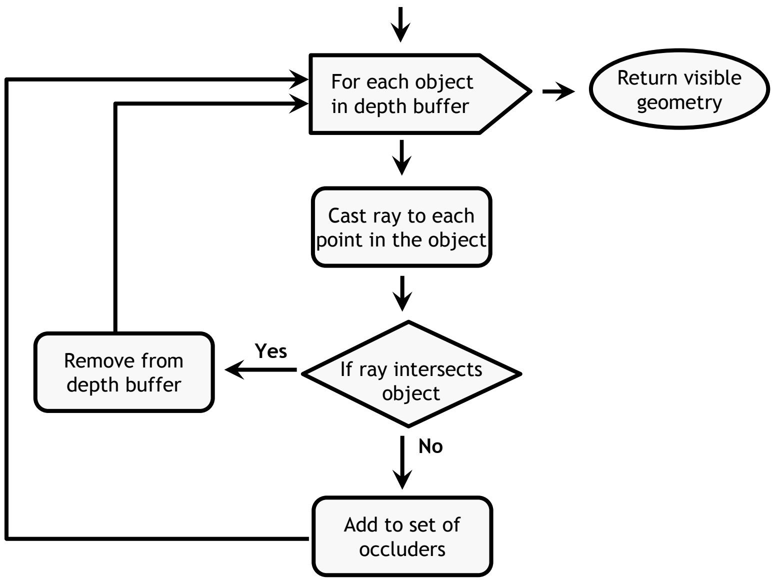


Figure 3.23: Ray casting visibility algorithm flow chart

casting, illustrated in Figure 3.22, is based on creating sight lines between the point of view and an object to be tested for visibility. If all of these sight lines (or rays) intersect with closer geometry, then the object is not visible. This algorithm is illustrated by a flow chart in Figure 3.23. Each object is taken in closest-first order from the depth buffer. Rays are cast to the corners of the object's bounding volume to test if they intersect any of the occluders. Occluders in this case are defined as any geometry closer to the point of view, as determined by the depth buffer. If all of these rays intersect some geometry between the object and the user's PoV, then the object is taken to be invisible and is removed from the depth buffer. Otherwise the object remains in the depth buffer as a possible occluder and the algorithm continues until the whole depth buffer has been traversed. The ray casting algorithm may identify visible objects as invisible due to the use of only four rays. Increasing the number of rays increases the accuracy of the algorithm at the cost of lower performance. It is expected that this approach will perform well in terms of accuracy and performance in environments that are not densely occluded as every object needs to be tested.

The protocol for choosing ray casting over backface culling, or vice versa, is dependent on the relative performance of the implementation of these algorithms, as the computational complexity of the visibility service impacts middleware responsiveness and energy consumption. An evaluation of the relative performance of these algorithms can be found in Chapter 5, Section 5.1.3.5.

There are a number of trade-offs made in determining object visibility between the accuracy of the solution and its computation complexity. The first design decision that impacts accuracy is the decision to have a binary, visible or not visible, object precision result. This indicates whether objects (that is, buildings, billboards, people, etc.) are visible and does not discern between the case where, for example, a corner of a building is visible versus the case where the entire front elevation of a building can be seen. The use of conservative culling via bounding volumes in order to reduce the number of geometric operations (intersection, containment, and distance) that need to be carried out also increases imprecision. There is no guarantee that an object will be visible if the bounding volume is partially visible, since the bounding volume encloses more space than the object itself [61]. The visibility service is designed for 2D geometry, further reducing accuracy as tall objects behind shorter occluding objects are not detected. 2.5D or 3D environment models have been used by others to address this issue using servers [132, 84]. All visibility determination is point-based as opposed to the region-based visibility proposed by Cohen-Or et al. [47]. Point-based visibility assumes that the user's position is known and does not take the imprecision of positioning systems into account. These trade-offs significantly reduce the computational complexity at the cost of reduced accuracy. The visibility determination, although not nearly accurate enough to render 3D scenes in computer graphics, is sufficient to provide a positive user experience of the types of visibility enabled services possible.

Despite these limitations, the visibility determination service has a number of advantages over other approaches. The use of a depth buffer in the occlusion culling algorithms results in the most important visible objects being found first. This enables the time taken to search for visible objects to be bounded, trading completeness of the visible set for predictable performance. This allows the performance of visibility determination to be adapted to suit hardware capabilities and still results in usable visible sets, particularly in urban environments that tend to be densely occluded (i.e., only a small portion of the environment is visible at any time) [47]. The visibility determination algorithms (view frustum, depth buffer and occlusion culling) are also designed to operate on-demand in dynamic environments. There is no requirement to perform any pre-computation or storing of visibility information, thereby reducing the memory requirements of the solution.

Object precision visibility determination is an example of a spatial service designed for mobile spatially-aware applications, using the environment model and limited set of spatial operations as building blocks (RQ-3). Visibility determination generates conservative estimates of the visible set of objects using specially adapted view frustum culling, depth buffers and occlusion culling. The algorithm is designed to minimise demand on the computational resources of hand-held mobile devices, maintaining application responsiveness and preserving energy.

3.4 Generic Framework

This section describes the design of a generic framework for spatially-aware mobile applications that incorporates the algorithms and services of thie current thesis into a reusable library of spatial tools. The framework is designed to address the common challenges posed by hand-held mobile devices in a reusable and extensible manner [110]. This reduces the cost and time required to build spatially-aware applications, by allowing developers to reuse and extend generic framework components.

The model reported here is generic in that it provides spatial services that are not confined to a single application domain, but may be reused by developers of a range of mobile applications. Reuse of the model is directly supported by a set of spatial services that are available to developers of spatially-aware applications (See Section 2.1). In addition, the MRDB is compatible with a common GIS data format, enabling the reuse of existing spatial data. Extensibility in the model is supported by the pluggable component model that defines specific extension points where alternative algorithms may be substituted at deployment time.

3.4.1 Pluggable Component Model

Figure 3.24 illustrates the pluggable component model that facilitates the run-time substitution of component implementations. This is achieved using a configuration file that specifies the component in use and its parameters. For example, the framework contains a visibility determination service (Section 3.3.6) that can use a number of visibility culling algorithms. The configuration file (Appendix A.1) specifies which of these algorithms are in use and what the parameters (if any) should be set. Having

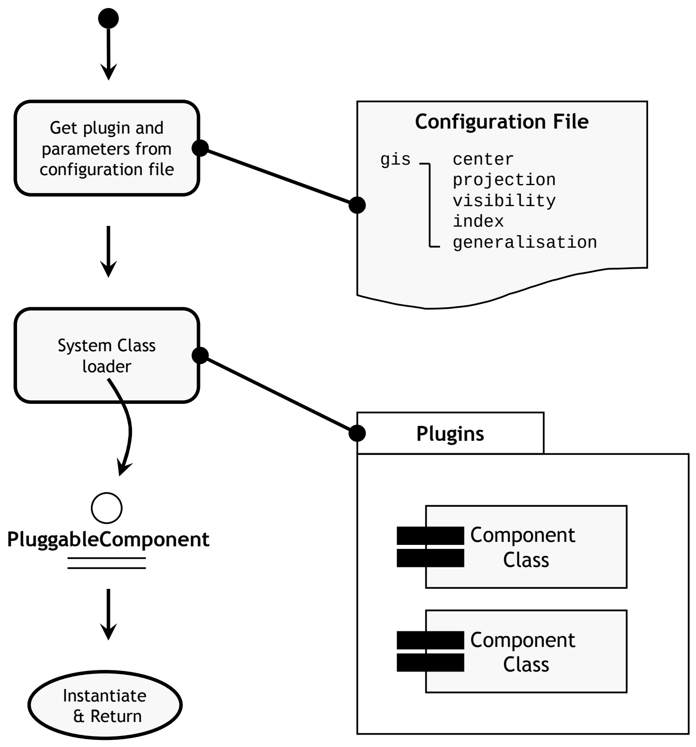


Figure 3.24: Pluggable component model

consulted the configuration file, the system class loader is used to create an instance of the component. In this way, changeability is built into the framework through parameterisation and extension points, where component implementations can be substituted. This also supports reuse of the framework as developers can can change components and parameters without recompiling code.

Figure 3.25 shows a component level architecture of the framework. The components providing the spatial services presented in Section 2.1 are shown. Cylindrical shapes indicate files, rectangles indicate components and dashed rectangles represent pluggable component interfaces. Dashed arrows indicate an “is a sub-component of” relationship.

The bottom layer of the library contains components supporting the reading and writing of spatial geometry, feature attributes and configuration data to and from files. Above the bottom layer there are two components: a *Spatial Index* (Section 3.2.2), and the MRDB (Section 3.2.1). The spatial index supports the retrieval of geometry from multiple underlying spatial data fragments. The MRDB allows geometry to be retrieved at any resolution without incurring the cost of computing a scale specific representation on each request.

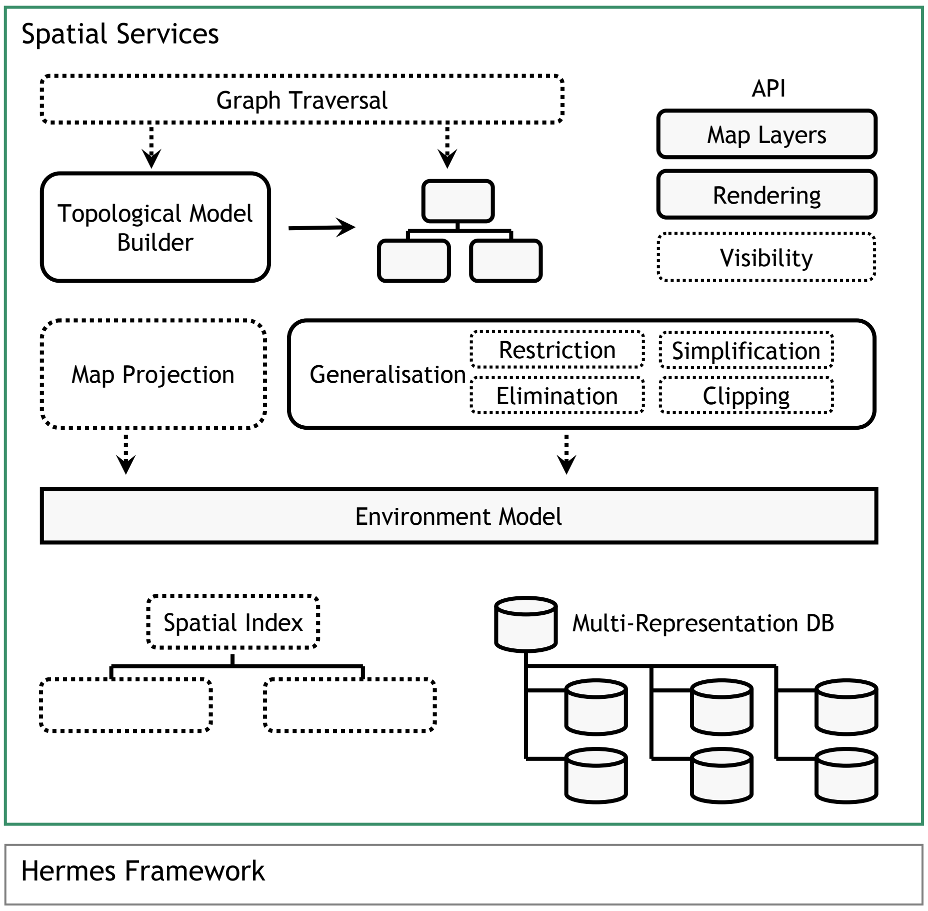


Figure 3.25: Pluggable component model extension points

The next layer in the architecture is responsible for adapting geometry to make it suitable for display at different resolutions (Section 3.2.3) and projecting from the underlying coordinate reference system of the spatial geometry to the mobile device's screen coordinate system (Section 3.3.4). The environment model provides access to spatially referenced data based on its real world extents, attributes and scale.

The top layer of the architecture contains a number of components that are designed to provide a simple and extensible interface to the framework. The *Rendering* component provides a means for developers to control the way cartographic representations are rendered (Section 3.3.2). The *MapLayers* and *Buffering* components provide the methods that allow dynamic information to be drawn on top of more static cartographic data. A *Route Generation* component exists that constructs a graph-based topological representation of the spatial data that is searched using one of a number of pluggable graph traversal algorithms (Section 3.3.5). A *Visibility* component provides a visibility determination service (Section 1.3.4).

The component-level extension points in the framework are illustrated in Figure 3.25 as components with a dashed outline. These pluggable components are all replaceable. Further extension of the framework is possible at the sub-component level, where options exist to implement alternative algorithms. These sub-component extension points are: the occlusion culling algorithm of the visibility service, graph traversal and edge weighting algorithms of the route generation service and the map projection algorithms of the coordinate transformation service. The layer design of the map rendering service further supports extension of the framework through inheritance.

3.4.2 Framework Summary

The framework for spatial services is a generic model for spatial services that addresses the common challenges posed by hand-held mobile devices in a reusable and extensible manner. The framework is decomposed into components, each responsible for providing a reusable spatial service. A pluggable component model provides a means to configure the framework through parameterisation.

3.5 Chapter Summary

The current chapter describes a model for spatial middleware that allows spatially-aware mobile applications to maintain a dynamic model of the users' environment locally. The environment model is represented in a multiple representation database that uses spatial data from a standard GIS data format and makes it available at different levels of detail, generalising the data on-demand. The multiple representation database is populated with spatial data by means of the collaboration facilities of the Hermes framework, which manage the acquisition and dissemination of spatial data in an ad-hoc wireless networking environment. The design of application-level spatial services including adaptable map rendering, spatial reasoning, coordinate transformation, route generation and visibility determination were also described. These services make use of the algorithms for accessing multi-scale spatial information to provide these middleware services on hand-held mobile devices without overburdening their limited resources or having to rely on wireless network connectivity. The chapter described the design of a generic framework for spatially-aware mobile applications, incorporating the algorithms and services already presented into a reusable library of spatial tools that can be reused and extended to reduce the cost of building spatially-aware applications. The design presented is optimised for efficient implementation on hand-held devices and features logically decomposed components that reuse design metaphors familiar to developers of spatially-aware applications. The following chapter presents the implementation detail of the spatial services and generic framework described in the current chapter.

Chapter 4

Implementation

The last chapter described the design of a spatial data model and a set of spatial operations for mobile spatially-aware applications. As described in this chapter, these components are combined to form a generic framework that maintains a model of the user's environment.

The framework, implemented in Java[1], is composed of generic implementations of environment modeling and spatial operations that can be reused by spatially-aware applications. This chapter describes in detail how the framework is implemented. The chapter begins with a high-level overview of the packages and components that comprise the application framework, including their dependencies and relationships. This is followed by a a presentation of the mobile platform and Hermes framework, describing the reuse of Hermes functionality and integration of Hermes services. The implementation of the multiple representation spatial data model that dynamically generates smaller scale representations via model-oriented generalisation is presented along with a number of the framework's spatial services. Finally, the implementation of the framework as a reusable library of spatial tools is presented, highlighting implementation features that facilitate reuse and extensibility.

It is a goal of the implementation, presented in this chapter, to produce a framework that is as lightweight as possible. All implementation choices support this goal by minimising memory usage and computational complexity to reduce energy consumption and improve responsiveness. These implementation choices range from language specific, method level code optimisations (reuse of instantiated objects, breaking loops

[1]Java 2 Micro Edition (J2ME) Personal Basis Profile [146]

early, avoiding expensive constructs, etc.) to system wide policies such as the use of caching to reduce repeated computation, the parameterisation of method fidelity and the trading of accuracy or efficiency. This chapter chooses not present code optimisations as it would require a line by line analysis and contributes little to the readers understanding of how the research questions are addressed. In contrast, where implementation policies have played a part, their effect is discussed.Architecture Overview

The model for spatial middleware is implemented as an application framework in Java, specifically, Java 2 Micro Edition (J2ME). J2ME was chosen for the following reasons:

1. *Executes on device.* Java software is stored on mobile devices, allowing it to be executed even if no network connection is present. This is a significant advantage over WAP or XHTML-MP based mobile applications, which are hosted on a HTTP server. The use of a software platform that executes on device contributes to the provision of spatial services on hand-held devices despite the inherent unreliability of wireless networks (RQ-1).

2. *Platform independent.* Software written in Java can be run on any device supporting this technology, eliminating the need to develop a different version for each mobile platform. This supports software reuse and reduces the cost of development of spatially-aware applications based on the framework (RQ-5).

3. *Availability.* Java is available on the majority of today's mobile devices. In addition, tools exist to support development, deployment and device configuration. Also, APIs and libraries are available that provide access to specialised hardware (for example, GPS sensors) [67].

4. *Performance.* The platform independence of Java is achieved by way of a platform independent byte-code representation that is interpreted by a virtual machine at run time. This process introduces overhead that traditional, non interpreted languages do not incur. However, the performance of modern Java virtual machines (JVM) is increasingly comparable to C, or C++ code [126, 8]. In addition, J2ME is designed specifically for resource-limited devices, supports minimal configurations of the JVM and includes limited Java APIs that contain only the essential capabilities for a mobile device.

5. *Ubiquity.* Java is used by many of today's spatially-aware applications. For example, Java is used by: the MultiMeetMobile Location-based Service [227]; Point-to-Discover [213]; GeoNotes [68]; ComMotion [136]; and Guide [43]. As spatially-aware applications are largely being developed using Java, a middleware framework for spatially aware applications can make the most significant contribution and be most easily integrated if developed in the same language.

4.0.1 Platform

Figure 4.1 illustrates the software and hardware platform on which the framework was implemented. Both the framework for spatial middleware and Hermes [10] were developed using the IBM WebSphere Everyplace Micro Environment (J9) JVM[2]. The IBM JVM supports the Personal Basis Profile of J2ME, which is designed for resource constrained devices like mobile phones and PDAs. It uses the Connected Device Configuration (CLDC) as its basis. The Pocket PC hardware used during development and evaluation was a HP IPAQ (h6300) device running Windows Mobile 2003 but the framework is designed to be platform independent and can be run on any comparably capable hardware.

4.0.2 Packages

Figure 4.2 shows an overview of the spatial middleware. The diagram is a UML package diagram with added dependency relationships and components. Shading is used to denote package hierarchy and is not intended to illustrate any grouping of packages.

The framework is implemented as a highly modular collection of components. Packages are used to decompose implementation classes into component-level groupings. At the top of the diagram an `Interface` package is shown to contain the `MapCanvas` component. This user interface component supports the inclusion of map-based interfaces into spatially-aware applications. The `MapCanvas` component is dependent on the `MapContext` component. The design of this component is illustrated as a class in

[2]The J9 VM is an implementation of the Java Virtual Machine Specification, Version 1.3. J9 VM can be downloaded as part of IBM Workplace Client Technology, Micro Edition 5.7 from the IBM website (requires registration).

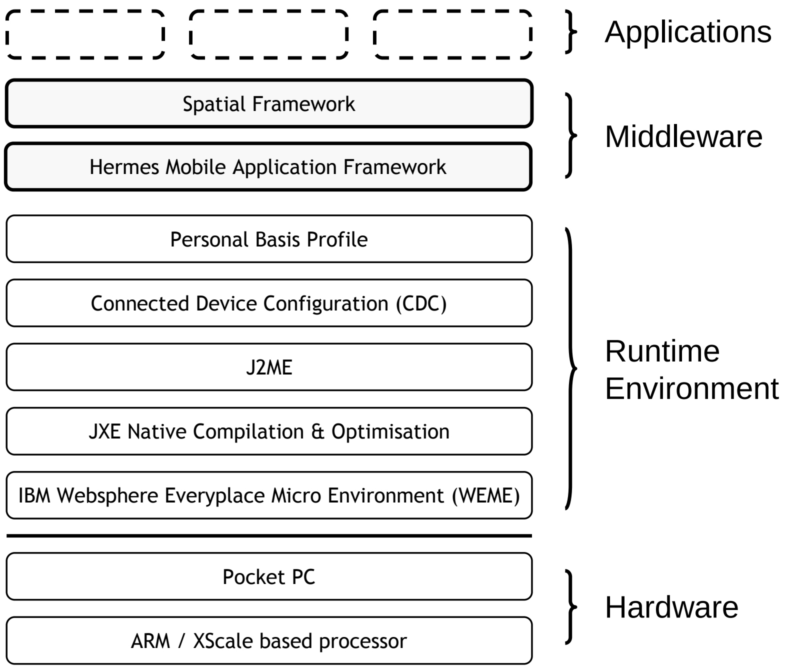

Applications
Spatial Framework
Hermes Mobile Application Framework
Middleware
Personal Basis Profile
Connected Device Configuration (CDC)
J2ME
JXE Native Compilation & Optimisation
IBM Websphere Everyplace Micro Environment (WEME)
Runtime Environment
Pocket PC
ARM / XScale based processor
Hardware

Figure 4.1: Implementation platform

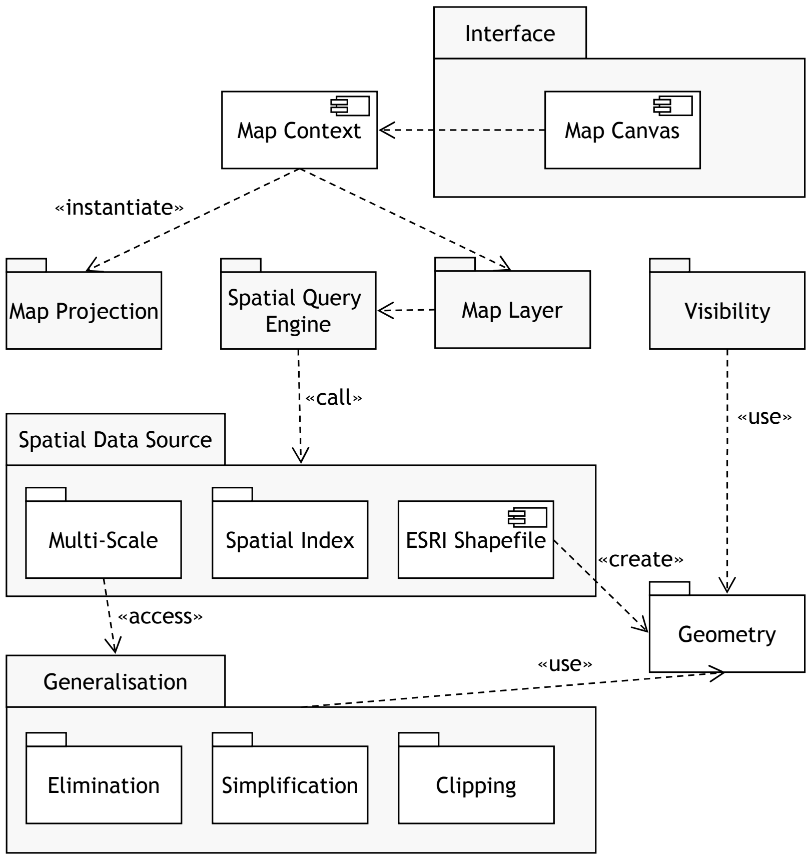


Figure 4.2: Architecture overview package diagram with dependencies

Section 3.3.2, Figure 3.12. The `MapContext` has access to `Map Layer` and `Map Projection` packages. The `Map Layer` package contains the abstract `Map Layer` design (Section 3.3.2, Figure 3.12) and a collection of implementations of specific map layers. For example: a `ShapeMapLayer`, that displays a single type of geometry specified by its attributes; `GraticuleMapLayer`, that draws a grid to help read coordinates; `CompassMapLayer`, that overlays a compass rose to indicate north on a rotatable map; and `OSMMapLayer`, a raster map layer that displays map tiles retrieved from the Open Street Map project[3]. `ShapeMapLayers` in the `Map Layer` package depend on a `Spatial Query Engine` package to do the work of retrieving `Geometry` objects from `Spatial Data Source` objects (Section 3.3.3, Figure 3.13). The `Spatial Data Source` package contains sub-packages for `Multi-Scale` data sources (Section 3.2.1), `Spatial Indexes` (Section 3.2.2) and an `ESRI ShapeFile` component. The `Multi-Scale` package contains the code necessary to implement the MRDB and depends on access to the `Generalisation` (Section 3.2.3) package to perform model-oriented generalisation. The `Generalisation` package makes use of the `Geometry` package and contains sub-packages for performing `Elimination` (Section 3.2.3.2), `Simplification` (Section 3.2.3.3) and `Clipping` (Section 3.2.3.4).

A complete class diagram is not included due to the scale of the code base. Where interesting implementation detail exists that can not be directly deduced from the design it is presented in the following sections.

The total size of the code base is illustrated in Figure 4.3, a pie-chart showing non-commenting source statements (NCSS) per package gathered using JavaNCSS [124]. Some of the minor sub-packages have been collapsed into their higher level packages to improve readability and packages containing benchmarking and testing code have been omitted. In total, the code base is decomposed into 32 packages, 203 classes, 1354 functions and has a total of 10,882 NCSS. The full output from the JavaNCSS program showing all packages can be found in the Appendix (Section B).

4.0.3 Hermes Integration

The model for mobile spatial middleware relies on the Hermes mobile application framework to support collaborative spatial data dissemination and context manage-

[3]OpenStreetMap - The Free Wiki World Map, `http://www.openstreetmap.org/`

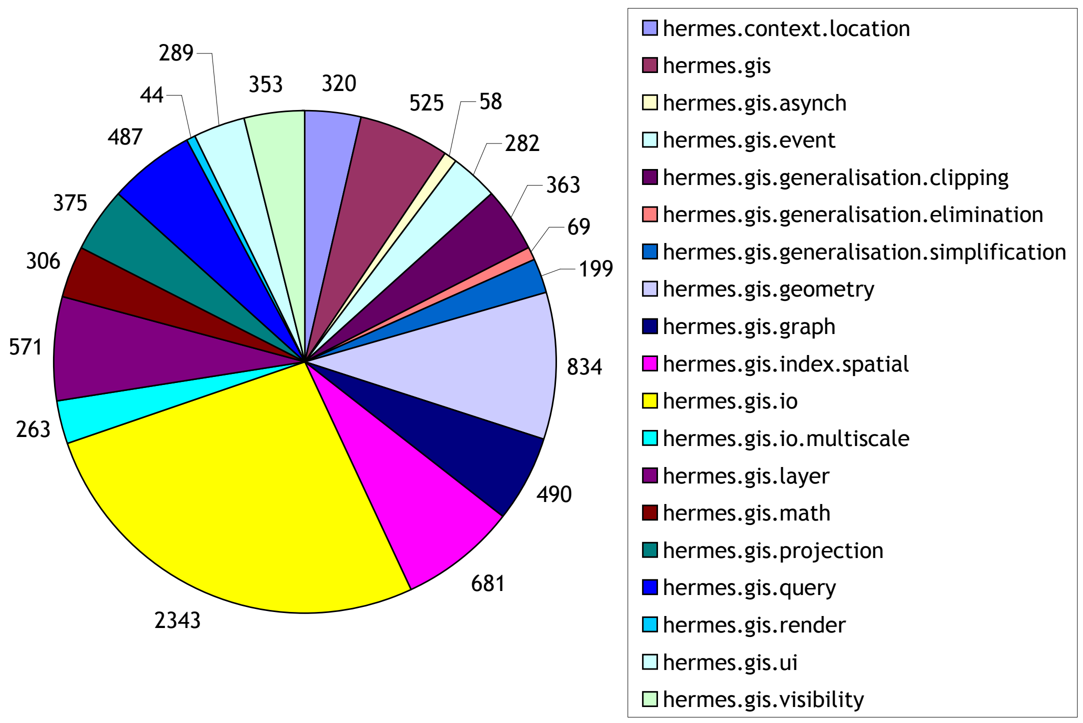


Figure 4.3: Non-commenting source statements per package

ment (Section 3.1.2). This functionality is used to acquire spatial data, eliminating any dependency on continuous network connectivity. In addition to collaboration, Hermes supports context management. These services are used by the spatial middleware to access information about the user's environment such as location and orientation.

This section describes the implementation detail of how the model for mobile spatial middleware integrates with Hermes and accesses the application frameworks services. We limit the discussion to the usage of Hermes in relation to configuration and integration with the spatial middleware[4].

To acquire spatial data using Hermes there are a number of configuration steps that must be taken. First, the Hermes configuration file has to be configured to support collaboration (Listing 4.1).

Listing 4.1: Hermes basic configuration parameters

```
# The Serial Comms API to use (J2ME)
hermes.comms.serial.provider=J2MESerialConnection
# Communication properties
hermes.comms.threads.incoming=2
```

[4]For Hermes framework implementation detail see Linehan & Spence [10] and Driver [63]

```
hermes.comms.heartbeattimeout=30000
hermes.comms.broadcast.port=4446
hermes.comms.broadcast.tick=10000
hermes.comms.listening.timeout=10000
hermes.comms.listening.tick=5000
hermes.comms.receive.timeout=5000
hermes.comms.receive.tick=5000
# GPS (IPAQ)
hermes.comms.gps.comm=7
hermes.comms.gps.baud=9600
# The Context Container to use
hermes.context.container.provider=DB4OContextContainer
hermes.context.container.file=data/db4o
# Debugging
hermes.debug.log=false
hermes.debug.log.file=data/hermes.log
```

Listing 4.1 shows an example of a basic Hermes configuration [10]. The important parameters to support collaboration are the `hermes.comms` properties. These parameters control the behaviour of the service discovery and communication components of the Hermes framework. Once the configuration is created the Hermes framework needs to be initialised and informed of any local sources of context (sensors) to which it should connect.

Listing 4.2: Instructing Hermes to use a local context source (GPS sensor)

```java
SerialAddress serialPort = new SerialAddress(commPort, baudRate);
// Class modeling sensor device
GPSReceiver gps = new GPSReceiver(serialPort);
// Get description of type of context provided by source
ContextServiceDescription[] sd = (ContextServiceDescription[])
    gps.getContextServiceDescriptions();
// Discover the new local context Source
Hermes.getCommunication().getServiceDiscovery().
    updateContextSourceList(gps);
```

Listing 4.2 shows the addition of a local context source to the Hermes framework. Hermes now knows about the context source, the type of context information it supplies and how to communicate with it. After executing these lines of code, Hermes still does not begin acquiring location information from the GPS receiver because it has not been told what types of context applications and higher level services require. In the case of the spatial middleware, spatial data, location and orientation context is required. The next step is to instruct the Hermes framework to acquire these types of context.

Listing 4.3: Acquiring and sharing context in Hermes

```
1 // acquire position for centering map
2 Hermes.setInterestInContextTypes(_mapContext, ContextType.
     LOCATION, 200);
3 // acquire orientation for rotating map
4 Hermes.setInterestInContextTypes(_mapContext, ContextType.
     ORIENTATION, 50);
5 // acquire spatial data
6 Hermes.setInterestInContextTypes(_gisDataSource, ContextType.
     GEOMETRY, 800);
7 // On the outbound side
8 Hermes.getPrivacy().addContextTypeToDisclose(ContextType.GEOMETRY
     , 5000);
```

Listing 4.3 shows the calls to the Hermes framework instructing it on which types of context information to acquire and share. `Hermes.setInterestInContextTypes()` takes three parameters: a reference to an object that is listening for events notifying it of a change in a particular context value, the type of context to acquire and the maximum frequency of context change notifications. Line 8 in the code listing is instructing the outbound side of the collaboration process, the Hermes privacy component, that it can disclose the context type `GEOMETRY` (that is, spatial data).

The Hermes framework can now begin to manage the collaboration with sensors, peer devices and infrastructure to acquire and share spatial data and positional information. This process happens without any additional input from the spatial middleware and is completely transparent to spatially-aware mobile application developers.

Listing 4.4: Retrieving location context from Hermes

```
// Ask Hermes for its location manager service (part of the
    context model)
LocationManager locMgr = (LocationManager) Hermes.
    getSystemService(Context.LOCATION);
// Get the current location
Location loc = null;
try {
        loc = locMgr.getLocation();
} catch (LocationException e) {
        MobileGIS.log.error(e.getMessage(), e);
}
// Get the locations coordinates
if (loc != null)
        Point p = loc.getCoordinates();
```

After Hermes has been configured, informed about local context sources and instructed to acquire specific types of context, the context information must then be retrieved. Listing 4.4 is an example of how the spatial middleware retrieves location information from Hermes. The code snippet gets a handle on the system service that supplies location context and then calls the `getLocation()` method. The latitude and longitude coordinates are then extracted from the `Location` object using the `getCoordinates()` method. The implementation of the system services for delivering specific context types is influenced by the The Open Handset Alliance[5] Android 1.0[6] application framework for mobile devices [90]. The Android architecture shares the notion of system services that provide access to types of context (including location).

4.1 Spatial Data Model

The model for spatial middleware acquires then stores spatial data locally. To provide a usable dynamic model of the user's environment, it is necessary to be able to make

[5]The Open Handset Alliance `http://www.openhandsetalliance.com/`
[6]Android `http://www.android.com/`

spatial data available in different resolutions and levels of detail to allow for flexible zooming on small displays. This is achieved with the MRDB, the design of which is described in Section 3.4.

This section presents the implementation of the MRDB in detail. We begin by describing the ESRI Shapefile open GIS data format and how it is extended to support spatial data at multiple resolutions. This is followed by a detailed implementation level view of the algorithm for retrieving spatial data at a specific resolution from the file components that constitute the extended Shapefile format. Next, the `QueryEngine` interface to the MRDB is presented. The `QueryEngine` provides an interface for accessing spatial data at multiple levels of detail independent of underlying data store. Finally, the implementation of the `Generalisation` package is presented.

The algorithms for accessing spatial data stored in the extended Shapefile format in conjunction with the `QueryEngine` interface and internal model-oriented generalisation algorithms contribute to the ability to manipulate spatial data on hand-held mobile devices, addressing RQ-2, the design of algorithms to access a spatial model at multiple levels of detail.

4.1.1 ESRI Shapefiles

ESRI Shapefiles store non-topological geometry and attribute information for spatial objects. The geometry for a feature is stored as a shape comprising a set of vector coordinates [69]. Shapefiles consists of a main file, an index file, and a dBase[7] table. The main file stores shapes with a list of its vertices. The index file contains the offset of the main file record from the beginning of the main file for each record. The dBase table contains feature attributes with one row per shape record in the main file.

Table 4.1 summarises the three file components of a Shapefile. Each Shapefile can contain only one type of geometry (that is, lines, areas or points) and Shapefiles do not have any notion of scale or resolution. To support the retrieval of spatial geometry at multiple levels of detail it is necessary to be able to store multiple copies of the same spatial objects at different resolutions. The Shapefile format can not do this and so has been extended.

[7]dBase was one of the first database management systems published in the early 1980s [49]. The underlying file format, the .dbf file, is used by many applications as a simple format to store structured data.

Extension	Description
SHP	The main file containing the geometry.
DBF	A dBase attribute database containing a row for each geometry object in the SHP file.
SHX	An index to SHP files. Contains byte offsets and content lengths for individual geometry objects in the SHP file.

Table 4.1: ESRI Shapefile files

Extension	Description
X.SHP	A SHP file, acting as a multi-resolution cache of the main file.
X.SHX	An index to the copy SHP file containing a re-mapping of record offsets for shape records.
QIX	An R-Tree spatial index.

Table 4.2: Extended ESRI Shapefile file components

4.1.2 Multi-Scale Shapefile

The ESRI Shapefile data format has been extended in two ways. Additional file components are added to store the multiple copies of spatial objects at different scales and a hierarchical spatial index is added to improve the performance of identifying shape records within a particular extent.

Table 4.2 lists the components that are added to the Shapefile standard. The first is a copy of the main (SHP) geometry file. This file is added to as objects are generalised on-demand from the main file. Theses generalised objects are written to the file sequentially in an order that may be different to their original order in the main file. To account for this reordering of the variable length shape records, a copy of the index to the geometry file is needed at each level of detail (LoD). Multiple pairs of SHP and SHX may be created depending on the number of LoDs. These files are associates with the original files through a simple file naming scheme. For each LoD, x, an SHP and SHX exists with the names *example.x.shp* and *example.x.shx* where the main file in the standard Shapefile is named *example.shp.* The QIX file is an R-Tree spatial index that provides access to the main file based on each spatial objects bounding volume. Querying this data structure results in a list of shape records that may then be read from the main SHP file.

In addition to adding files to the standard, the file headers are modified. The SHP and SHX files contain a 100 byte header. This header contains information such as,

Position	**Field**	**Type**
Byte 0	File Code	Integer
Byte 4	*Min. Scale*	*Double*
Byte 12	*Max. Scale*	*Double*
Byte 24	File Length	Integer
Byte 28	Version	Integer
Byte 32	Shape Type	Integer
Byte 36	Bounds	Bounding Box

Table 4.3: Extended Shapefile main file header

the version of the format, length of the file, type of spatial data contained (point, line or area) and bounding volume of the data.

Of the 100 bytes in the file header there is a group of 20 unused bytes reserved for future features. These 20 bytes are used to store two double (8 byte) values representing the scale range of the data stored within the file. Table 4.3 shows the fields in the modified file header with their byte position, value and type.

4.1.3 MRDB Spatial Data Retrieval

The extended Shapefile format contains three base files containing geometry, attributes and an index (See MRDB Shapefile components: Figure 3.5 on page 69). For each LoD, there are two additional files replicating the structure of the main geometry and index files but containing a portion of the records at a lower resolution and in a different order. The algorithm for locating, reading and returning a spatial object at a specific scale from this collection of files is illustrated in the sequence diagram, Figure 4.4.

The `QueryEngine` begins by retrieving the type of geometry from the main files header and matching it with the type of spatial data requested in the `SpatialQuery`. The index is then queried with the `getRecords()` call which takes a spatial extent as a parameter and returns list of shape records. The DBF file is then searched for the attributes that belong to those records and these values are matched against any specified attributes in the `SpatialQuery`. The list of shape records is then passed to an elimination filter which removes records that will not be visible when rendered on the device screen.

At this stage, the `QueryEngine` begins the process of reading the actual geometry records from the SHP file. This process begins by identifying the LoD containing

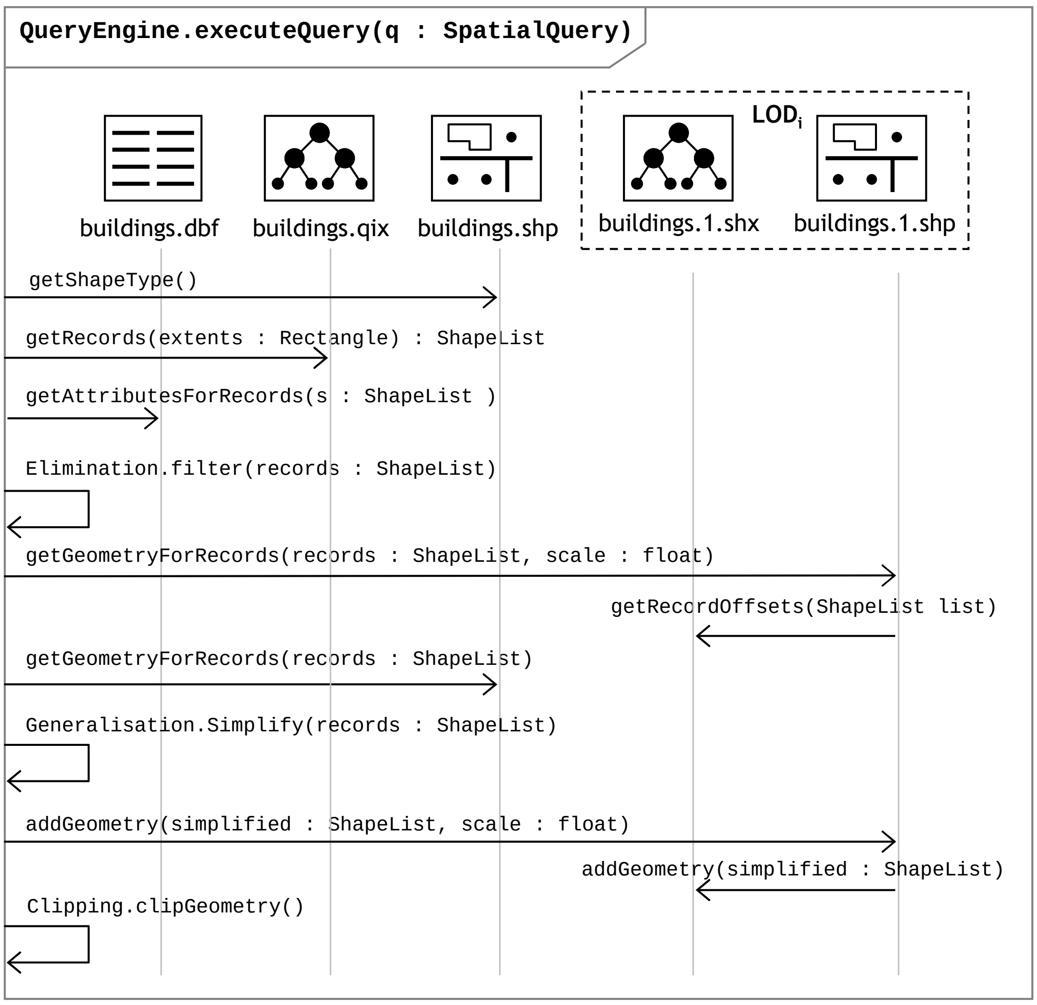

QueryEngine.executeQuery(q : SpatialQuery)
LOD$_i$
buildings.dbf
buildings.qix
buildings.shp
buildings.1.shx
buildings.1.shp
getShapeType()
getRecords(extents : Rectangle) : ShapeList
getAttributesForRecords(s : ShapeList)
Elimination.filter(records : ShapeList)
getGeometryForRecords(records : ShapeList, scale : float)
getRecordOffsets(ShapeList list)
getGeometryForRecords(records : ShapeList)
Generalisation.Simplify(records : ShapeList)
addGeometry(simplified : ShapeList, scale : float)
addGeometry(simplified : ShapeList)
Clipping.clipGeometry()

Figure 4.4: Multi-scale retrieval

geometry records at the correct scale. `getRecordOffsets()` is called to re-map the record offsets. This step is necessary as the order of the entries in each of the standard files in the Shapefile is synchronized but the copies of these files at each LoD contain the same records in a different order and size. If geometry has been found for all the records then the algorithm continues from `Clipping.clipGeometry()`. Otherwise, the remaining geometry is read from a higher resolution LoD (or the main file if no multi-scale copies exist). The `getGeometryForRecords(records : ShapeList)` method is called for each of the LoDs until the main file is reached. The geometry is generalised to the correct scale and added to the appropriate LoD to prevent the same objects from being generalised again. Once complete the `QueryEngine.executeQuery()` returns a list of spatial objects including their geometry and attributes at the requested scale.

A number of features of the multi-scale retrieval algorithm contribute to making it as computationally lightweight as possible. The matching of attributes, geometry type and elimination are performed on an objects minimum bounding volume, retrieved from the spatial index. This reduces the number of objects for which geometry has to be retrieved, minimising the amount of I/O and the need for generalisation. Generalisation is only performed when data for a spatial object is not available in the correct LoD cache. In this case, data for the object is retrieved from the closest LoD and further generalised to minimise the impact of the generalisation process. Once geometry is generalised it is written back to a persistent LoD cache. All reading and writing to memory is buffered to minimise the number of system calls. Finally, clipping is used to eliminate points from spatial objects only where the majority of the object is outside the queries spatial extents. This prevents the clipping of small numbers of points from shapes, where the processing time saved by the reduction in points is less than the processing required to perform the clipping.

4.1.4 Spatial Query Interface

The implementation features a spatial query interface that provides a convenient interface to the complex MRDB format. Figure 4.5 is a class diagram detailing the implementation of the spatial query interface.

A `QueryEngine` is responsible for executing queries. A query is specified programmatically and is modeled by the `Query` interface. This interface is realised by the

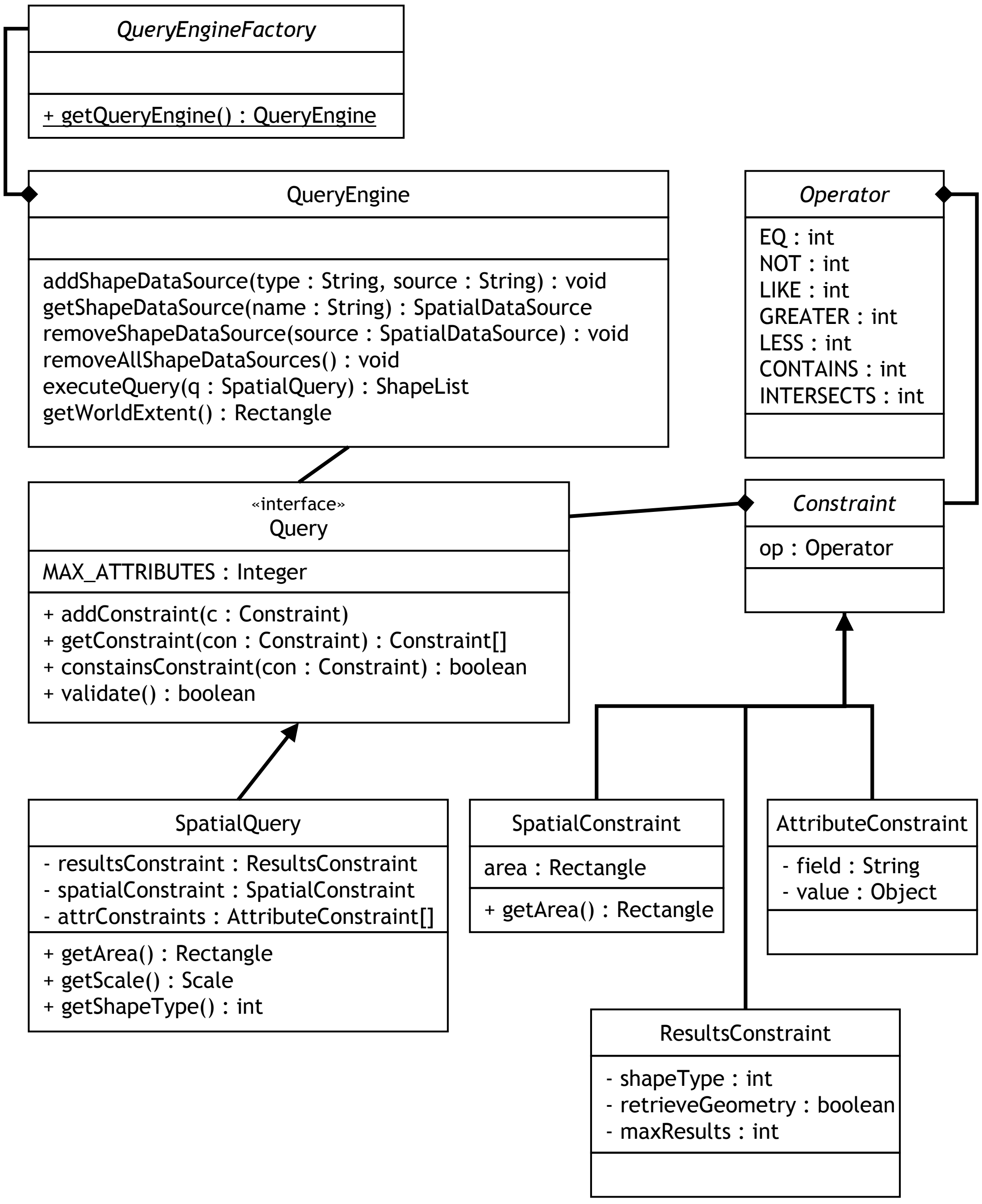

QueryEngineFactory
+ getQueryEngine() : QueryEngine
QueryEngine
addShapeDataSource(type : String, source : String) : void
getShapeDataSource(name : String) : SpatialDataSource
removeShapeDataSource(source : SpatialDataSource) : void
removeAllShapeDataSources() : void
executeQuery(q : SpatialQuery) : ShapeList
getWorldExtent() : Rectangle
Operator
EQ : int
NOT : int
LIKE : int
GREATER : int
LESS : int
CONTAINS : int
INTERSECTS : int
«interface»
Query
MAX_ATTRIBUTES : Integer
+ addConstraint(c : Constraint)
+ getConstraint(con : Constraint) : Constraint[]
+ constainsConstraint(con : Constraint) : boolean
+ validate() : boolean
Constraint
op : Operator
SpatialQuery
- resultsConstraint : ResultsConstraint
- spatialConstraint : SpatialConstraint
- attrConstraints : AttributeConstraint[]
+ getArea() : Rectangle
+ getScale() : Scale
+ getShapeType() : int
SpatialConstraint
area : Rectangle
+ getArea() : Rectangle
AttributeConstraint
- field : String
- value : Object
ResultsConstraint
- shapeType : int
- retrieveGeometry : boolean
- maxResults : int

Figure 4.5: Spatial query class diagram

SpatialQuery class, which supports three types of query Constraint: SpatialConstraint, ResultsConstraint and AttributeConstraint. The SpatialConstraint is required by the SpatialQuery class. It specifies the the bounding area for which spatial information is required. The SpatialConstraint, like all other classes realising the abstract Constraint class, contains an Operator object. The ResultsConstraint class adds extra constraints to the spatial query such as, the type of geometry (point, line or area) and the maximum number of results to return. This constraint also contains a boolean, retrieveGeometry that is used to indicate to the QueryEngine whether geometry should be returned or just a list of shape records. The AttributeConstraint class constrains the possible values of a particular field in a spatial objects attributes. Multiple AttributeConstraint instances can be added to a SpatialQuery.

Listing 4.5: Building a spatial query

```
SpatialQuery sq = new SpatialQuery(new SpatialConstraint(
    MapContext.getProjection().getWorldExtent()));
sq.addConstraint(new ResultsConstraint(ShapeConstants.
    SHAPE_TYPE_POLYGON, 1600));
sq.addConstraint(new AttributeConstraint("Layer", "COUNTRY",
    Operator.EQ));
```

Building an instance of the SpatialQuery class is illustrated in Listing 4.5. This query specifies an area (MapContext.getProjection().getWorldExtent()) in which polygons should be retrieved at scale 1600 where their "Layer" attribute is equal (EQ) to "COUNTRY". A text-based representation of this same query is shown in Listing 4.6. This SQL-like representation of spatial queries is implemented for debugging purposes only and is not directly executed, rather the programmatic version is passed to the QueryEngine (Figure 4.5).

Listing 4.6: Text representation of a spatial query

```
SELECT polygons WHERE BOUNDS CONTAIN Rectangle: [4] {(53.340572,
    −6.2515793) (53.342297, −6.2515793) (53.342297, −6.2486973)
    (53.342297, −6.2486973) } AND Layer = 'COUNTRY';
```

4.1.5 Generalisation

Model-oriented generalisation provides the MRDB with the ability to automatically derive new LoDs on demand. The design chapter has presented the four generalisation operations that the spatial middleware implements: restriction, elimination, simplification and clipping (Section 3.2.3). Restriction is the elimination of spatial objects from spatial query results based on their attributes. Restriction is implemented by the spatial query interface, which allows attribute constraints to be specified; the query engine which retrieves objects from the MRDB and interprets the spatial queries; and the Layer-based map rendering components, which support the overlay of multiple collections of geometry that are restricted by their types and attributes.

The implementation of elimination, simplification and clipping has not yet been presented and so will be detailed in the following sections. These generalisation steps are implemented in a modular, extendable manner, making use of interfaces to allow for the extension of the framework through the addition of alternative implementations of these components.

4.1.5.1 Elimination

The elimination generalisation step (Section 3.2.3.2) is implemented as a filter to the `ShapeList` class (Figure 4.6). The method `accept(r : ShapeRecord) : boolean` is called on each `ShapeRecord` in the `ShapeList`. If the method returns *false*, then it is removed from the list.

The `EliminationFilter` is an abstract class that implements the `ShapeListFilter` interface and uses the Singleton design pattern [123]. The Singleton pattern is used to ensure that there is only ever one instance of an object realising the `EliminationFilter` abstract class definition. This is part of the pluggable component model, the implementation of which is presented in Section 4.3.1.

The `MinimumAreaFilter` is the realisation of the `EliminationFilter` component. This object maintains a minimum real world area that is visible on the screen. This is used as a threshold. Spatial objects whose total area is below this value are removed from the list. The threshold value is configurable through a properties file and can be used to remove objects too small to be visible or just to declutter the interface by removing less important objects.

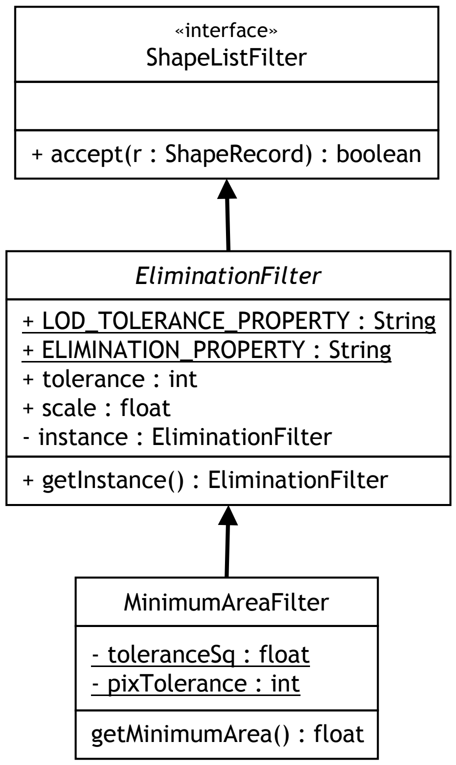

«interface»
ShapeListFilter
+ accept(r : ShapeRecord) : boolean
EliminationFilter
+ LOD_TOLERANCE_PROPERTY : String
+ ELIMINATION_PROPERTY : String
+ tolerance : int
+ scale : float
- instance : EliminationFilter
+ getInstance() : EliminationFilter
MinimumAreaFilter
- toleranceSq : float
- pixTolerance : int
getMinimumArea() : float

Figure 4.6: Elimination `ShapeList` filter class diagram

4.1.5.2 Simplification

The simplification generalisation step reduces the complexity of individual shapes producing information suitable for display at different levels of detail (Section 3.2.3.3). Simplification is implemented as a component, defined by the `Simplify2D` interface (Figure 4.7). The interface has methods that take a 2D shape and a scale as parameters and return a simplified version of the shape. The concrete `ShapeSimplify` class realises the `Simplify2D` interface. Internally, the `ShapeSimplify` class has private instances of two algorithms for reducing the complexity of 2D geometry, `KraakOrmeling` and `DouglasPeuker`. The algorithm instances also implement the `Simplify2D` interface and are loaded as pluggable components at run time. Two algorithms are implemented because of their different characteristics. The `KraakOrmeling` class uses the *nth* point algorithm that simply removes every *nth* point from the shape. This is computationally fast but can cause distortion to shapes that contain few points. The `DouglasPeucker` class uses a more sophisticated algorithm that minimises the distortion of the shape but is more computationally expensive [62]. The `ShapeSimplify` class uses `DouglasPeuker` only in situations where shapes have few points and for

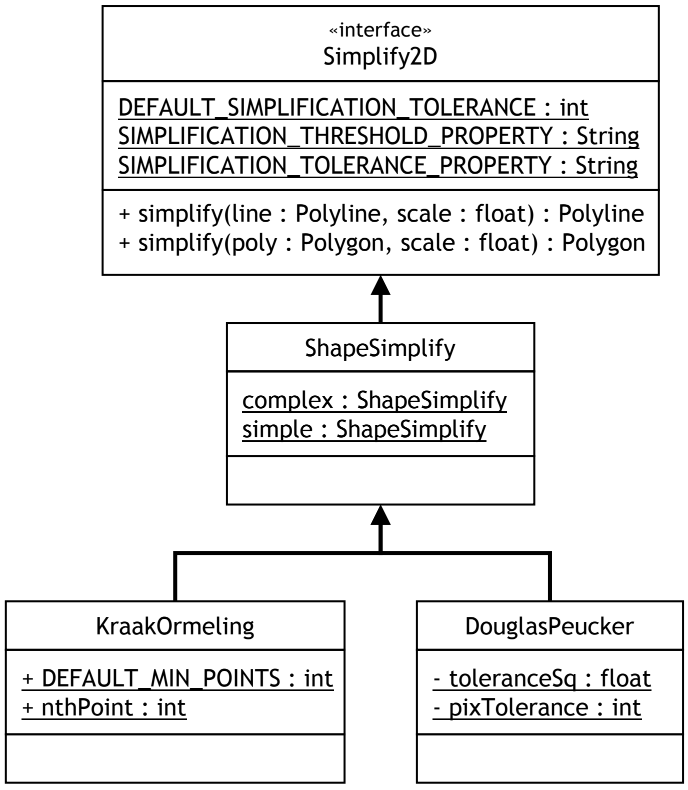

«interface»
Simplify2D
DEFAULT_SIMPLIFICATION_TOLERANCE : int
SIMPLIFICATION_THRESHOLD_PROPERTY : String
SIMPLIFICATION_TOLERANCE_PROPERTY : String
+ simplify(line : Polyline, scale : float) : Polyline
+ simplify(poly : Polygon, scale : float) : Polygon
ShapeSimplify
complex : ShapeSimplify
simple : ShapeSimplify
KraakOrmeling
+ DEFAULT_MIN_POINTS : int
+ nthPoint : int
DouglasPeucker
- toleranceSq : float
- pixTolerance : int

Figure 4.7: 2D geometry simplification class diagram

scale values that would expose any distortion in the shape.

4.1.5.3 Clipping

Clipping is the modification of individual shapes to remove portions that extend beyond the viewport of the device, to preserve computational resources that are wasted computing projections and rendering coordinates that will not be seen (Section 3.2.3.4). Figure 4.8 is a class diagram illustrating the implementation of clipping. Clipping algorithms are designed either for line clipping or polygon clipping. Because of this, the implementation features three interfaces: `Line2DClip` (line clipping), `Polyline2DClip` (polyline clipping), and `Polygon2DCLip` (polygon clipping). Implementations of clipping algorithms realise these interfaces making them interchangeable and the implementation extensible. These implementations are:

- `CohenSutherland` - Code taken from "Computer Graphics for Java Programmers" by Ammeraal [9] and altered to support the geometry model (Chapter 3, Figure 3.13) and `Polyline2DClip` interface.

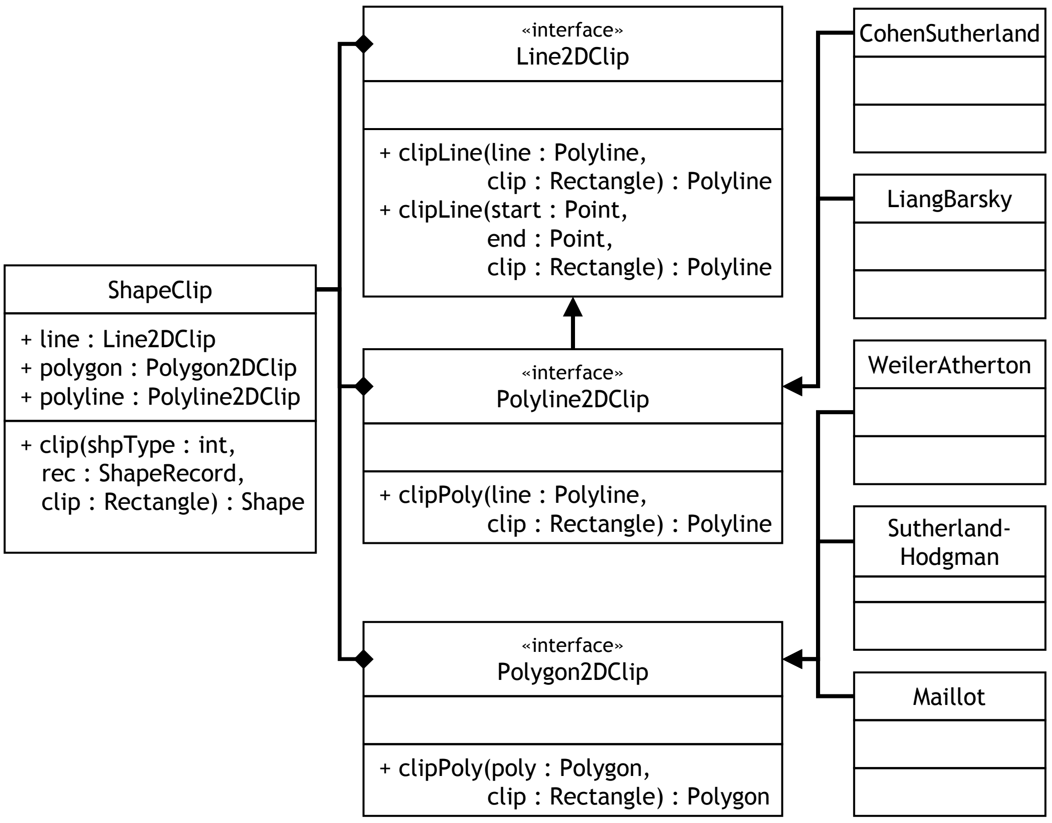

«interface»
Line2DClip
+ clipLine(line : Polyline,
clip : Rectangle) : Polyline
+ clipLine(start : Point,
end : Point,
clip : Rectangle) : Polyline
CohenSutherland
LiangBarsky
ShapeClip
+ line : Line2DClip
+ polygon : Polygon2DClip
+ polyline : Polyline2DClip
+ clip(shpType : int,
rec : ShapeRecord,
clip : Rectangle) : Shape
«interface»
Polyline2DClip
+ clipPoly(line : Polyline,
clip : Rectangle) : Polyline
WeilerAtherton
Sutherland-
Hodgman
«interface»
Polygon2DClip
+ clipPoly(poly : Polygon,
clip : Rectangle) : Polygon
Maillot

Figure 4.8: Geometry clipping class diagram

- `LiangBarsky` - Ported from a C# implementation by Grishul Eugeny[8].
- `SutherlandHodgman` - Implementation based on pseudocode for polygon clipping posted online by Huiling Yang [242].
- `WeilerAtherton` - An optimised version of the Weiler-Atherton algorithm for rectangular clipping [115]. Implemented using an algorithm described by William Shoaff[9].
- `Maillot` A port of a C implementation published by Maillot [133].

A `ShapeClip` class provides an interface to the clipping functionality to the rest of the framework, while making the details of the actual implementing algorithm transparent to other components. The only method of this class is `clip(shpType : int, rec : ShapeRecord, clip : Rectangle)`, which takes as parameters the type of shape

[8]Wikipedia, Liang-Barsky, Revision: January 2008 `http://en.wikipedia.org/wiki/Liang-Barsky`.

[9]Clipping, Last Updated: March 2002, `http://www.cs.fit.edu/~wds/classes/graphics/Clip/clip/clip.html`

that is to be clipped (which is used to identify a suitable clipping algorithm), the shape itself and the clipping rectangle.

4.2 Spatial Services

The model for mobile spatial middleware presented in this thesis incorporates support for a number of common application-level spatial services. The previous chapter has detailed the design of spatial services including adaptable map rendering (Section 3.3.2), spatial reasoning (Section 3.3.3), coordinate transformation (Section 3.3.4), route generation (Section 3.3.5) and visibility determination (Section 3.3.6).

This section presents the implementation of these spatial services for mobile spatially-aware applications using the 2D environment model described in Section 3.1. The design chapter has already detailed an object-oriented design for these spatial services (Section 3.3). As the implementation platform is Java, an object-oriented language, the implementation of these services follows directly on from the design. As the implementation of many of the services is a direct mapping of the object-oriented design to code it is not necessary to describe the implementation in detail as its structure and algorithms can be found in the design.

One service that warrants further discussion is the visibility determination service as its implementation can not easily be derived from the description in Section 3.3.6.

4.2.1 Visibility Determination

The spatial visibility service determines the visibility of real world objects from the user's point of view using the 2D environment model presented in Section 3.1. Handheld mobile devices require a combination of collaborating components to determine the visibility of spatial objects on-demand in a dynamic scene. These components are:

1. `ViewFrustumCulling`: A component implemented as a filter for a list of spatial objects that eliminates objects that are either: 1) not in the direction the user is looking (as estimated by orientation sensors, that indicate the direction the user is facing or moving); or 2) too far away from the user to be visible.

2. `DepthBuffer`: A component that extends a shape list with an ordering based on distance from a specified PoV.

3. `OcclusionCulling`: A component that takes objects remaining after view-frustum culling ordered by a depth buffer and uses either a back-face culling or ray casting algorithm to remove objects that are occluded.

4.2.1.1 View Frustum Culling

View-frustum culling eliminates objects outside a FoV or beyond a defined distance from the user from a list of spatial objects (Section 3.3.6.1). A model of human visual perception (Figure 3.19, right) is used to determine the FoV and a cut off point beyond which objects should not be considered visible. View-frustum culling is implemented as a list filter, specifically the class `ViewFrustumCulling` realises the `ShapeList-Filter` interface and is passed as a parameter to the `ShapeList.filter(filter : ShapeListFilter)` method. The algorithm implemented for view-frustum culling is best illustrated by analysing the implementation of the `accept(Shape)` method that determines whether an element of a list should be removed or not (Listing 4.7).

Listing 4.7: The `accept()` method of the `ViewFrustumCulling` filter

```
public boolean accept(ShapeRecord record) {
        // If we don't know where user is looking assume they can
            see for 360 degrees around their location
        if (this._fieldOfView == 360) {
                // Check within viewing distance
                if (this._viewingPoint.distance(record.index.
                    bounds.getCenter()) > VisualPerceptionModel.
                    URBAN_FAR_POINT_METERS) {
                        return false;
                } else
                        return true;
        } else {
                // Check is in direction the user is looking
                float dir = this._viewingPoint.azimuth(record.
                    index.bounds.getCenter());
                if (dir > ((this._direction - (this._fieldOfView
                    / 2.0)) % 360) && dir < ((this._direction + (
                    this._fieldOfView / 2.0)) % 360)) {
                        // Check within distance user can see
                        if (this._viewingPoint.distance(record.
                            index.bounds.getCenter()) >
                            VisualPerceptionModel.
                            URBAN_FAR_POINT_METERS) {
                                return false;
                        } else
                                return true;
                }
        }
        return false;
}
```

The `accept(Shape)` method begins by checking the FoV. The FoV is measured in degrees, so a value of 360 indicates that the user's orientation is not known. In this case the algorithm has to assume that objects within a visible distance, irrespective of direction could potentially be visible. If the user's orientation is known, a FoV is determined based on the `VisualPerceptionModel`. The azimuth (or direction in degrees) from the user to the centre of the object being tested for visibility is calculated. The algorithm then tests to see if this value falls within the bounds of the user's FoV. If the object is within the user's FoV then the distance from the user is measured and if it is beyond a threshold defined by the `VisualPerceptionModel` (Chapter 3, Section 3.3.6.1) the object is marked as not visible.

4.2.1.2 Depth Buffer

The second important component is the depth buffer, implemented in the `DepthBuffer` class that extends and overrides a `ShapeList`. Figure 4.9 is a class diagram illustrating the implementation of the `DepthBuffer`. The depth buffer is essentially just a list of shapes that have been sorted by depth (that is, their distance from the user's location). Its implementation extends `ShapeList` and has a `DepthComparator` that is passed to

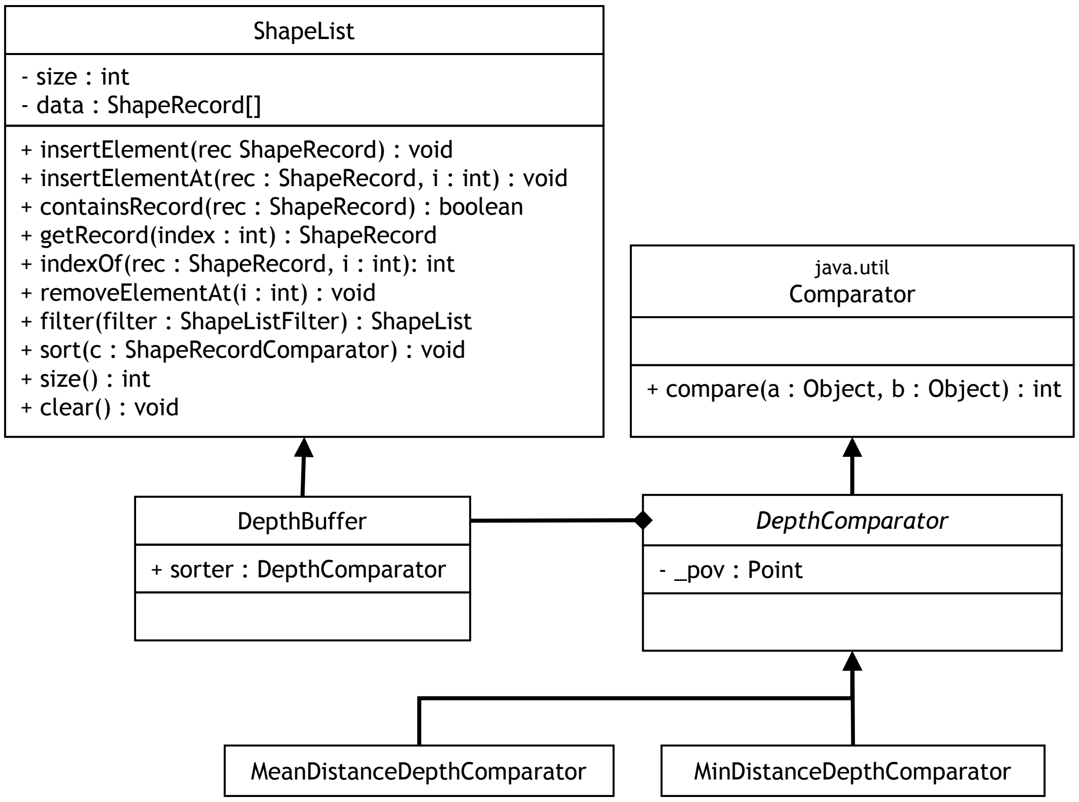

ShapeList
- size : int
- data : ShapeRecord[]
+ insertElement(rec ShapeRecord) : void
+ insertElementAt(rec : ShapeRecord, i : int) : void
+ containsRecord(rec : ShapeRecord) : boolean
+ getRecord(index : int) : ShapeRecord
+ indexOf(rec : ShapeRecord, i : int): int
+ removeElementAt(i : int) : void
+ filter(filter : ShapeListFilter) : ShapeList
+ sort(c : ShapeRecordComparator) : void
+ size() : int
+ clear() : void
java.util
Comparator
+ compare(a : Object, b : Object) : int
DepthBuffer
+ sorter : DepthComparator
DepthComparator
- _pov : Point
MeanDistanceDepthComparator
MinDistanceDepthComparator

Figure 4.9: Class diagram for `DepthBuffer`

the `ShapeList`'s `sort()` method. The `DepthComparator` is an abstract class to allow for multiple approaches to determining the depth of objects in a scene. Two approaches are implemented: `MeanDistanceDepthComparator` and `MinDistanceDepthCompara-tor`. The distinction between the two algorithms is illustrated in the design chapter (Figure 3.20). The `MeanDistanceDepthComprator` uses the distance from the centre of an object's bounding volume to the PoV to determine depth (for example, `depth = pov.distance(shape.bounds.getCenter());`). The `MinDistanceDepthComparator` uses the distance between the PoV and the closest point in the shape as its depth. The `DepthBuffer` overrides all of the `ShapeList`'s methods that modify the contents or order of the list to call `sort(dc : DepthComparator)` after each modification[10]. The sort method uses Java's built-in array sorting functionality (`Arrays.sort()`) to re-order the elements of the list based on the `DepthComparator`.

[10]Overridden methods call the super classes implementation and then re-sort the list.

4.2.1.3 Occlusion Culling

Occlusion culling is the process of filtering a set of objects to remove objects that are occluded by other objects (Section 3.3.6.3). An abstract `VisibilityCulling` class provides a factory method for instantiating an object that realises the `VisibilityCulling` interface. The actual class that gets created is defined by the configuration file. Two possible approaches have been implemented that realise this class, `BackfaceCulling` and `OcclusionCulling`.

Figure 4.10 is a UML activity diagram illustrating the implementation of the `BackfaceCulling` algorithm. The method that performs the visibility culling is named `occulsionCull(db : DepthBuffer)` and is defined in the `VisibilityCulling` interface. This method takes a `DepthBuffer` as a parameter and removes objects from the depth buffer that are not visible. The activity diagram is the implementation of the algorithm whose design is illustrated as a flow diagram in Figure 3.21.

The activity diagram shows that the algorithm iterates over all the objects in the depth buffer, starting at the closest object. For each object the angles to the two extreme visible edges are calculated. A triangular occlusion shadow is then created and added to any existing shadows. Starting with the next furthest away object, the remaining elements of the depth buffer are scanned to see if they are contained within the boundary of the occluding shadow. These objects are removed from the depth buffer because they can not be visible. The algorithm then returns to the next deepest object and continues until only visible objects remain.

The other class that implements `VisibilityCulling` is the `RayCasting` class. The ray casting algorithm is illustrated by a flow chart in Figure 3.23. Its implementation uses the intersection methods that are part of the geometry model to locate objects in the depth buffer to which no line of sight exists. Figure 4.11 is a UML activity diagram illustrating the implementation of ray casting occlusion culling. The implementation iterates over the objects in the depth buffer and calls the `isVisible(s : Shape)` method for each one. Objects that are not visible are removed from the depth buffer. The implementation of the `isVisible` method is illustrated separately. This method iterates through every point in the shape being tested. For each point a line is drawn from the PoV to the point. If the line intersects any geometry closer to the PoV (easily determined by looping over objects before this one in the depth buffer) then there is

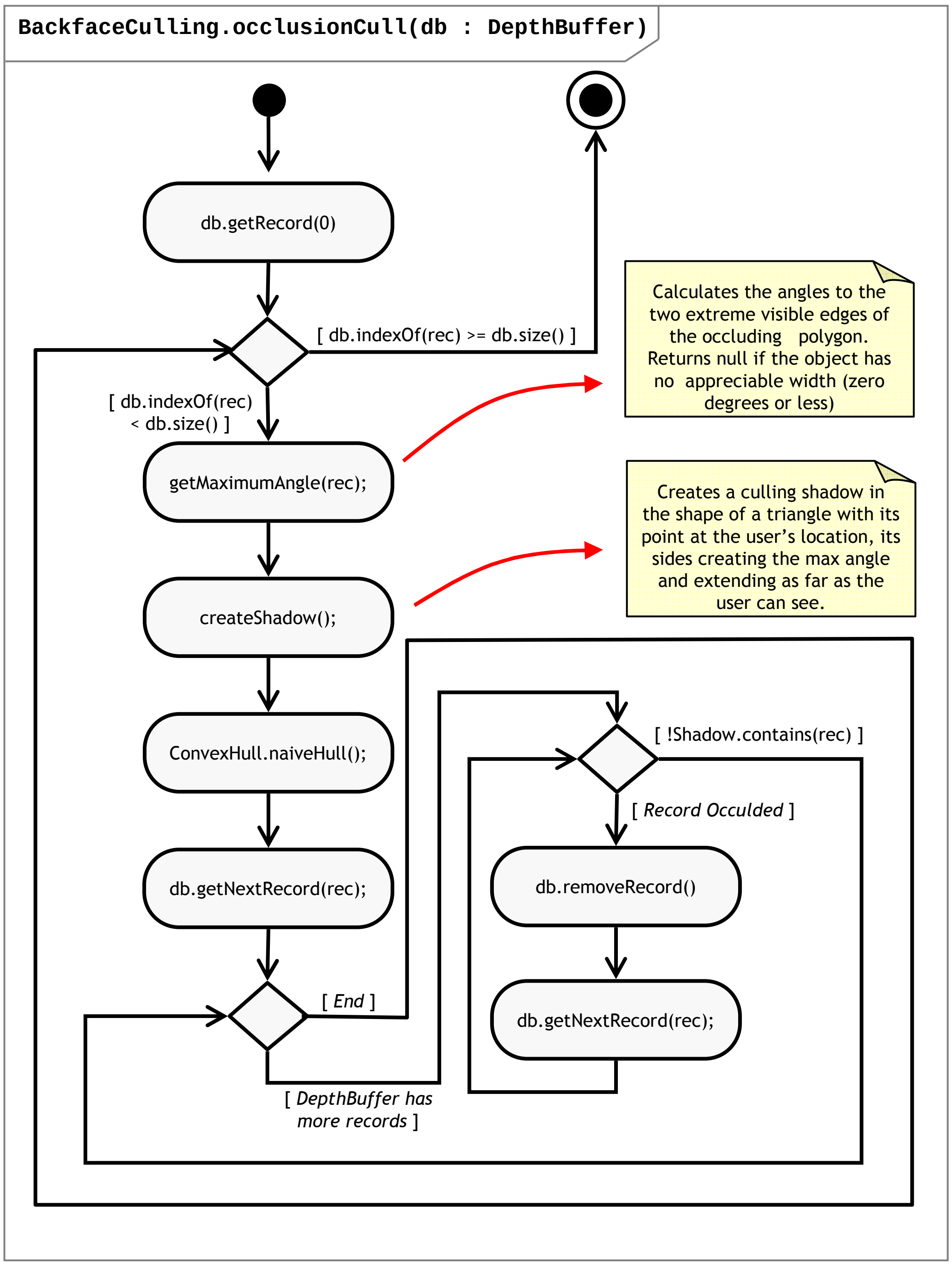

BackfaceCulling.occlusionCull(db : DepthBuffer)
db.getRecord(0)
[db.indexOf(rec) >= db.size()]
[db.indexOf(rec) < db.size()]
getMaximumAngle(rec);
Calculates the angles to the two extreme visible edges of the occluding polygon. Returns null if the object has no appreciable width (zero degrees or less)
createShadow();
Creates a culling shadow in the shape of a triangle with its point at the user's location, its sides creating the max angle and extending as far as the user can see.
ConvexHull.naiveHull();
db.getNextRecord(rec);
[!Shadow.contains(rec)]
[Record Occulded]
db.removeRecord()
db.getNextRecord(rec);
[End]
[DepthBuffer has more records]

Figure 4.10: Backface occlusion culling algorithm activity diagram

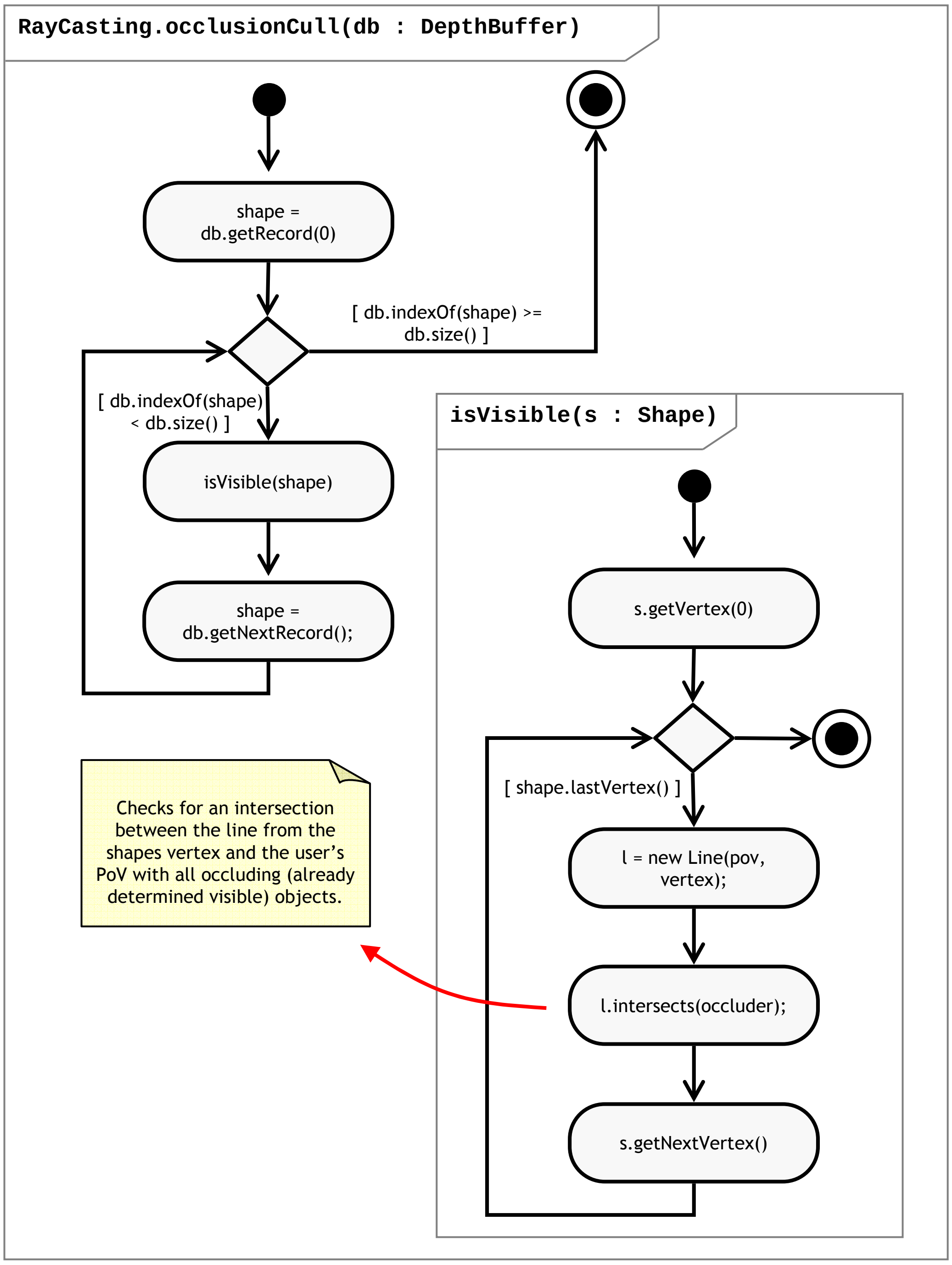

RayCasting.occlusionCull(db : DepthBuffer)
shape = db.getRecord(0)
[db.indexOf(shape) >= db.size()]
[db.indexOf(shape) < db.size()]
isVisible(shape)
shape = db.getNextRecord();
Checks for an intersection between the line from the shapes vertex and the user's PoV with all occluding (already determined visible) objects.
isVisible(s : Shape)
s.getVertex(0)
[shape.lastVertex()]
l = new Line(pov, vertex);
l.intersects(occluder);
s.getNextVertex()

Figure 4.11: Ray casting visibility determination activity diagram

no line of sight to that particular point. If none of the points in the shape can be seen then the shape itself is not visible and will be removed from the list.

4.3 Framework

This section describes the integration of the data structures, algorithms and spatial services into a generic framework for spatially-aware mobile applications (Section 3.4). The framework addresses the common challenges posed by hand-held mobile devices in a reusable and extensible manner. Many of the implementation choices presented so far contribute to the extensibility & reusability of the framework. For example:

- The choice of a platform independent development language that is supported by the majority of today's mobile devices, is already used by developers to build spatially-aware mobile applications and supports easy integration with the Hermes mobile application framework contributes to the reusability of the framework (Section 4.0.1).

- The decomposition of the framework into modular components, each responsible for providing a reusable spatial service and each reusing familiar mobile design patterns, metaphors and concepts (for example, the Android [90] influenced location context service (Section 4.0.3), the Open Geospatial Consortium [171] influenced geometry model (Section 3.3.3) and BBN Open Map [21] inspired the map layer model (Section 3.3.2).

- A pluggable component model that provides a means to configure the run time framework through parameterisation (Section 3.4.1).

- The multi-resolution spatial data model that has been implemented by extending an open standard GIS data format (Section 4.1).

- The simple spatial query model incorporating a human readable representation and object-oriented programming model (Section 4.1.4).

The most significant implementation features of the framework that contribute to its extensibility and reusability are the pluggable component model and persistent configuration files. The implementation of these features is presented in the following sections.

4.3.1 Pluggable Component Model

The pluggable component model defines specific extension points where alternative component implementations may be substituted at run time. The pluggable component model supports extensibility by enabling the addition of new algorithms and component implementations by simply modifying a configuration parameter.

Listing 4.8: The system class loader being invoked to instantiate a component

```
1  String codeBase = "ie.tcd.cs.dsg.hermes.gis.projection.";
2  String component = MobileGIS.getProperty(Projection.
       PROJECTION_PROPERTY);
3  // The constructor arguments for projections
4  Class[] projConstructorArguments = new Class[] {Point.class,
       float.class, int.class, int.class };
5  Object[] parameters = new Object[] { center, scale, width, height
       };
6  try {
7          this._projection = (Projection) Class.forName(codeBase +
               component).getConstructor(projConstructorArguments).
               newInstance(parameters);
8  } catch (InstantiationException e) {
9          MobileGIS.log.error(e.getMessage(), this);
10 // Additional exception handling omitted
11 }
```

Listing 4.8 illustrates how component interfaces are used to allow a component realising the interface to be specified in a properties file and dynamically loaded at run time[11]. The variable, `codeBase`, on line 1 specifies the package location of the component implementations. Line 2 retrieves the name of the class implementing the component from the frameworks configuration. The `Class.forName(implementation`

[11] The full extent of the error handling code is not shown.

: `String`) method instructs the JVM system class loader to load the specified code. Parameters (also specified in the configuration) are passed to the instance of the component.

Table 4.4 lists the seven locations in the framework implementation where pluggable components are used. The *Component Interface* defines the methods that the component must implement. The *Factory Class* is responsible for instantiating component implementations. Where the factory and component interface are listed as the same class, it is because the component interface is implemented using an abstract class that both defines the method that must be implemented in subclasses and contains a factory method capable of loading a component instance as specified by the configuration file (See Listing 4.8). The *Code Base* is the name of the package where component implementations are located and the *Configuration Property* is a key that is used to look up the component implementation class name in the configuration file (Section 4.3.2). The *Implementing Classes* are lists of the concrete component implementations included in the framework.

4.3.2 Configuration File

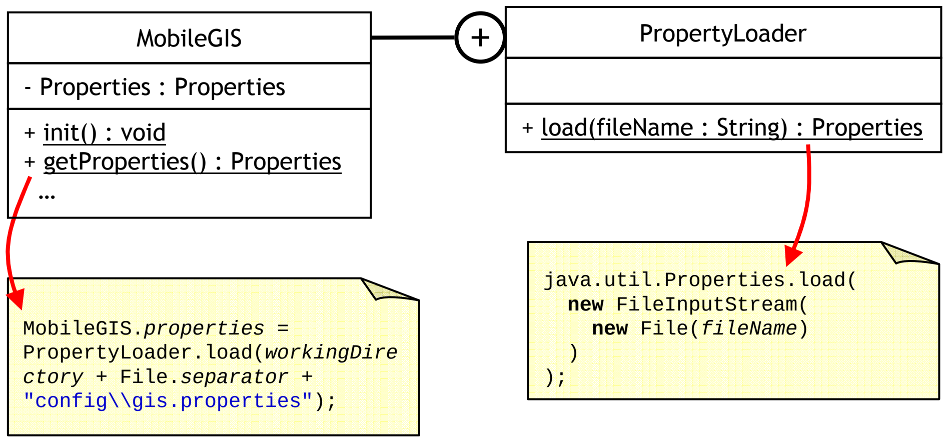
The configuration file allows pluggable component implementations to be specified and static parameters to be loaded at run time, customising the behaviour of the application framework without requiring source code modification. It is implemented using the `java.util.Properties` class. This class loads and stores key/value pairs from a file and manages them in memory, thereby facilitating the use of persistent configuration variables.

The framework has a single properties file that contains all its configuration (Appendix A.1). The properties are grouped by service and a naming scheme based on component package names is used. The configuration property groups are:

- `Debugging`; Parameters to control logging of activity during development and testing.
- `Networking`; Parameters used to specify a server address for a network raster map tile layer.
- `Screen`; The default mobile device screen dimensions.

Component Interface	Factory Class	Code Base	Configuration Property	Implementing Classes
VisibilityCulling	VisibilityCulling	hermes.gis.visibility	gis.projection.visibility. occlusion	BackfaceCulling, OcclusionCulling, MultiVisibilityCulling
DepthComparator	DepthBuffer	hermes.gis.visibility	gis.projection.depth	MeanDistanceDepthComparator, MinDistanceDepthComparator
Projection	MapContext	hermes.gis.projection	gis.projection	Mercator, EquiRectangular
EliminationFilter	EliminationFilter	hermes.gis.generalisation. elimination	gis.generalisation. elimination	MinimumAreaFilter, NullFilter
Line2DClip	ShapeClip	hermes.gis.generalisation. clipping	gis.projection.clip.line	LiangBarsky, CohenSutherland
Polygon2DClip	ShapeClip	hermes.gis.generalisation. clipping	gis.projection.clip.poly	WeilerAtherton, SutherlandHodgman, Maillot
Simplify2D	ShapeSimplify	hermes.gis.generalisation. simplification	gis.generalisation. elimination.simple, gis.generalisation. elimination.complex	DouglasPeucker, KraakOrmeling, Lang

Table 4.4: Pluggable components


MobileGIS
- Properties : Properties
+ init() : void
+ getProperties() : Properties
...
PropertyLoader
+ load(fileName : String) : Properties

```
MobileGIS.properties =
PropertyLoader.load(workingDire
ctory + File.separator +
"config\\gis.properties");
```

```
java.util.Properties.load(
  new FileInputStream(
    new File(fileName)
  )
);
```

Figure 4.12: Configuration loading

- `Scale`; The initial scale at which spatial data should be displayed on a map-based interface.
- `Centre of Map`; The default centre of the map. Used when a location context is not available.
- `Spatial Data Sources`; Specifies initial locally cached spatial data files.
- `Projection`; Specifies a pluggable component implementation to use.
- `Generalisation`; Specifies component implementation and parameters for generalisation algorithms.
- `Spatial Indexing`; Specifies configuration parameters for R-Tree spatial indexing and index persistence.
- `Visibility Algorithms`; Specifies component implementations for visibility determination algorithms.

The configuration key/value pairs are accessed at run time through a static call to the `MobileGIS getProperties()` method (Figure 4.12). This method uses a static inner class, `PropertyLoader`, to read the properties from a text file when the framework is first loaded. Once loaded the configuration is held in memory and written back each time a property is changed.

The properties in the configuration file are extensible by augmenting the existing files with new property specifications (no notion of extension via inheritance exists in relation to configuration files). For example, defining a new property involves adding the new property to the configuration file using the same notation as the existing properties. This can be done by modifying the file by hand or pragmatically using the `java.util.Properties.setProperty(key : String, value : String)` method.

4.4 Chapter Summary

This chapter described the implementation of a generic framework as a reusable library of spatial tools for maintaining a model of the user's environment and providing spatial services based on this model to mobile spatially-aware applications. This generic framework, the design of which is described in Chapter 3, provides reusable implementations of spatial services that do not overburden the limited resources of mobile devices and are not dependent on network connectivity.

The chapter presented the J2ME platform on which the framework is built. The implementation of integration mechanisms to allow the ad-hoc collaboration features of the Hermes framework to be used to maintain a dynamic model of the user's environment has been described. The implementation of the spatial data model including a detailed explanation of extensions to the Shapefile standard, the algorithm for retrieving spatial data from the file components of the proposed multi representation Shapefile, the spatial query interface and the generalisation steps have been described. The implementation of the visibility determination spatial service has been described along with a presentation of the implementation decisions that contribute to the extensibility & reusability of the framework.

Throughout the chapter, steps taken to ensure the implementation is as lightweight as possible are highlighted. These implementation policies, such as the use of caching to reduce repeated computation, the parameterisation of method fidelity and the trading of accuracy or efficiency, minimise memory usage and computational complexity.

The next chapter evaluates the performance of the frameworks spatial services and presents the results of experiments investigating the energy trade-offs in performing spatial operations locally versus off-loading the tasks to a server. A case study spatially-aware application demonstrates the reusability of the framework.

Chapter 5

Evaluation

The previous chapter described the implementation of a generic framework for maintaining a model of the user's environment and providing spatial services to mobile spatially-aware applications. This chapter describes performance evaluations, energy consumption evaluations and a case study. These evaluations assess the extent to which the framework satisfies the research questions (Section 1.2) in the following ways:

1. Spatially-aware mobile applications can maintain a dynamic model of the user's environment despite the inherent unreliability of wireless networks and limitations of hand-held mobile devices using the model for spatial middleware presented in this thesis (RQ-1). This model consists of an approach to collaboratively distributing spatial data containing meaningful map entities, such as streets, buildings and dynamic objects, in an interpretable format to mobile devices (Section 3.1.2 and Section 4.0.3). Once located on a mobile device, a multiple representation database (MRDB) based on an extended GIS data format uses the spatial data to maintain an environment model (Section 3.2.1 and Section 4.1). This approach addresses the challenges of research question one and is further reinforced in this chapter through the performance evaluation of the spatial services built on the locally maintained, dynamic model of the user's environment.

2. Chapter 3 shows how algorithms can be designed to access a spatial model at multiple levels of detail on hand-held devices that are limited in terms of battery power, processing resources and memory (RQ-2). The current chapter presents

evidence of the efficiency of these algorithms through performance and energy evaluations. Performance evaluations demonstrate that the algorithms execute fast enough on commercially available hand-held mobile devices to support interactive applications. Energy evaluations show the total energy consumption, CPU usage and memory usage of a simple spatially-aware application using these algorithms to access a spatial model at multiple levels of detail.

3. The design of application-level spatial services for mobile spatially-aware applications (RQ-3) are detailed in Section 2.1. This chapter demonstrates that these spatial services function as described through screenshots of an interactive spatially-aware application taken from a mobile device and through benchmarking their performance, demonstrating that they perform fast enough to meet user performance expectations.

4. The energy trade-offs in balancing computation with communication for this class of spatially-aware mobile application (RQ-4) are illustrated through comparative empirical experiments. These experiments show that the approach to maintaining a dynamic model of the user's environment locally, thereby eliminating the dependency on network connectivity, is more energy efficient than the more common approach of off-loading spatial operations to a server.

5. A generic model for middleware-based spatial services that address the common challenges posed by hand-held mobile devices (RQ-5) is provided by the framework described in Section 3.4 and Section 4.3. The current chapter presents an evaluation of the reusability and extensibility of the framework through the development of a case study application.

The remainder of this chapter is organised as follows. Section 5.1 evaluates the performance of individual spatial services of the framework, demonstrating that mobile devices can provide these services without relying on a wireless network connection. Section 5.2 presents an empirical experiment to determine the energy trade-offs in performing spatial operations locally versus off-loading the tasks to a server. Section 5.3 evaluates the genericity of the framework to support the development of a range of mobile, spatially-aware applications through the development of a case study application. The chapter concludes in Section 5.4 with a summary of the findings.

5.1 Spatial Service Performance

Evaluating the framework in relation to the research questions (Section 1.2) requires a demonstration that spatially-aware mobile applications can maintain a dynamic model of the user's environment (RQ-1), algorithms can be designed to access the environment model at multiple levels of detail (RQ-2) and application-level spatial services can be designed using the model (RQ-3). All these features must be possible without overburdening the limited resources of mobile devices, without relying on a constant network connection, and while maintaining the user experience of interactive mobile applications. To provide evidence that these features have been developed as described in the previous chapters, screenshots have been taken of a spatially-aware application with an interactive map-based interface[1] and their performance has been benchmarked. By benchmarking the performance of these services it is possible to verify that the algorithms execute fast enough on commercially available, hand-held mobile devices to support interactive spatially-aware applications.

This section quantifies the performance of the spatial middleware services through benchmarking. Acceptable bounds on performance are defined based on a user's tolerance for delay in an application [166]. Application response time literature suggests that there is a timescale within which it is optimal to deliver a result to the user. Returning an answer beyond this threshold has the potential to frustrate the user [195, 159]. Miller was the first to propose a set of guidelines for system response time [147]. He suggested a maximum delay of 2 seconds following a request. More recently, Nielsen concluded that a response time up to 10 seconds is now perceived as acceptable [167]. This thesis adopts a threshold of between 0-12 seconds as being a reasonable response time. This figure is based on the responsiveness requirements defined for other non-spatial services of the Hermes application framework [63].

This section begins with a presentation of the method used to benchmark spatial services. This is followed by the results themselves and concludes with an analysis of the results.

[1]Screenshots are included in Appendix A.4, Figures A.1, A.2, A.3 and A.4.

5.1.1 Methodology

There a number of challenges to measuring the performance of spatial services. Firstly, interpreted code is dynamically optimised at run-time. At start-up, the Java virtual machine (JVM) typically spends some time "warming up". During this period, methods may be compiled and optimised into native code [220]. This results in differing performance figures between the first time a block of code is executed and subsequent executions of the same block of code.

The timing system in Java also introduces “jitter” into measurements as the granularity of the timing mechanism can be coarser than the variability in benchmark timing. For example, on certain platforms the `System.currentTimeMillis()`[2] call has an effective 15ms granularity [111]. In addition to these Java specific challenges, various operating system activities may interfere with measurements by introducing CPU, memory, disk or network resource contention.

Finally, it is necessary to guage the trade-off between the use of sampling or an instrumented approach to gathering data. Sampling is supported by the JVM but can only identify specific bottlenecks and the sampling rate can introduce bias. Instrumented approaches can provide more data but can also impact the data gathered as instrumentation is compiled into the code being evaluated.

This thesis takes an instrumentation approach to performance evaluation. To ensure instrumentation code has a minimal effect on the spatial services, a benchmarking tool was developed (Figure 5.1). It is the responsibility of this tool to automate evaluations so that they may be repeated many times to avoid observations being biased by the “warming up” effect. The benchmarking tool also produces statistical reports based on instrumented code blocks during execution.

Figure 5.1 is a UML class diagram showing the Java benchmark harness. This tool is written as a set of non-extendable static classes to facilitate their compilation to native code at run-time. The main class is the `Harness` which is responsible for creating instances of `Benchmark` implementations and executing them[3]. The implementation of the abstract `Benchmark` will instrument some service of the framework with calls to the `Reporter`. The `Reporter` keeps track of timers on behalf of the `Benchmark`.

[2]Newer JVMs have a `System.nanoTime()` method that is more precise.

[3]The execution of `Benchmark` implementations by the `Harness` is listed in Appendix A.2.

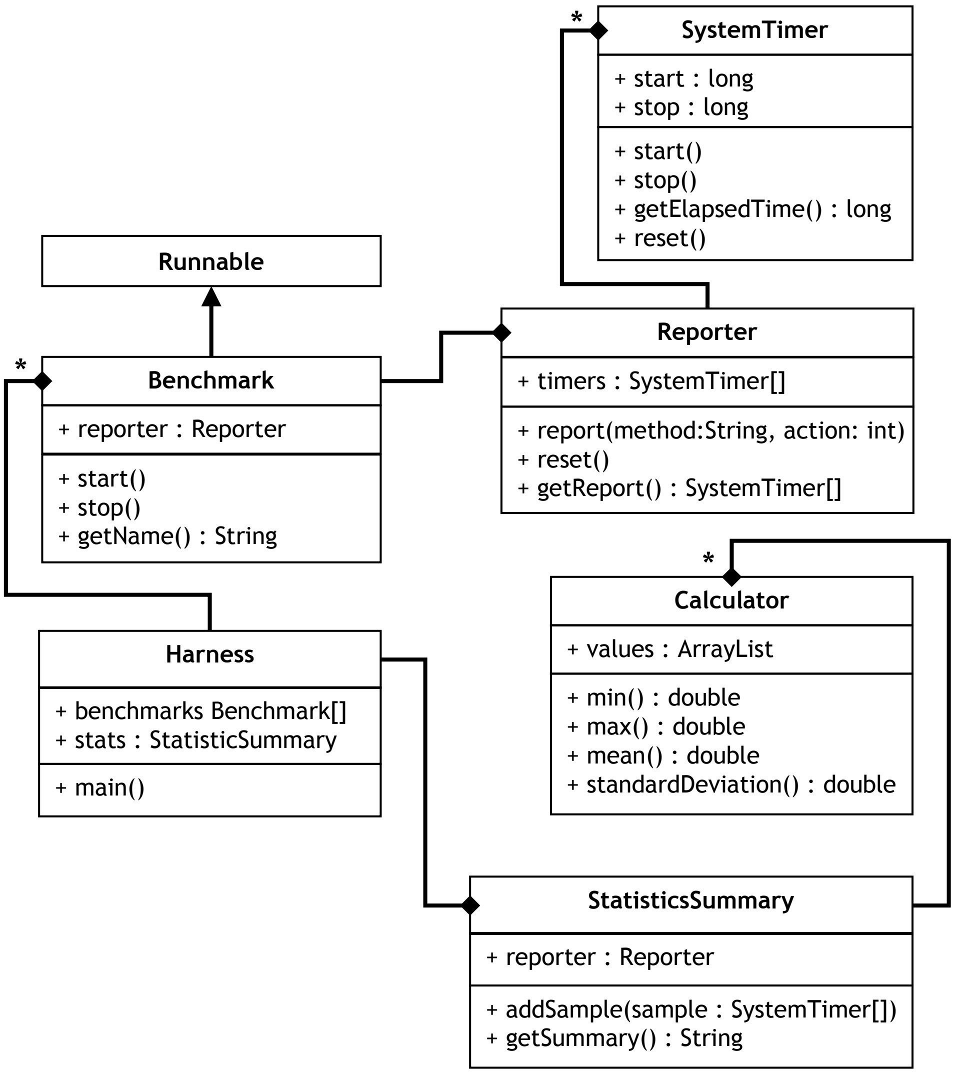


Figure 5.1: Benchmark harness for performance evaluations

After each execution of the benchmark the values of the timers are added as a sample to the `StatisticsSummary` by the `Harness`. On completion of the `Harness`, the `StatisticsSummary` class calculates a statistical summary of the samples and prints this data to a log.

The ability of the benchmark harness tool to minimise the impact of warm-up and jitter on performance measurements is illustrated by plotting the mean and standard deviation of a set of samples collected by the tool (Figure 5.2). To generate this graph, the same benchmark test was executed repeatedly. On each execution, its performance was recorded as a single sample. The mean and standard deviations of this set of samples were then calculated. The graph shows that the mean performance improves quickly as the number of iterations of the test increases. This demonstrates

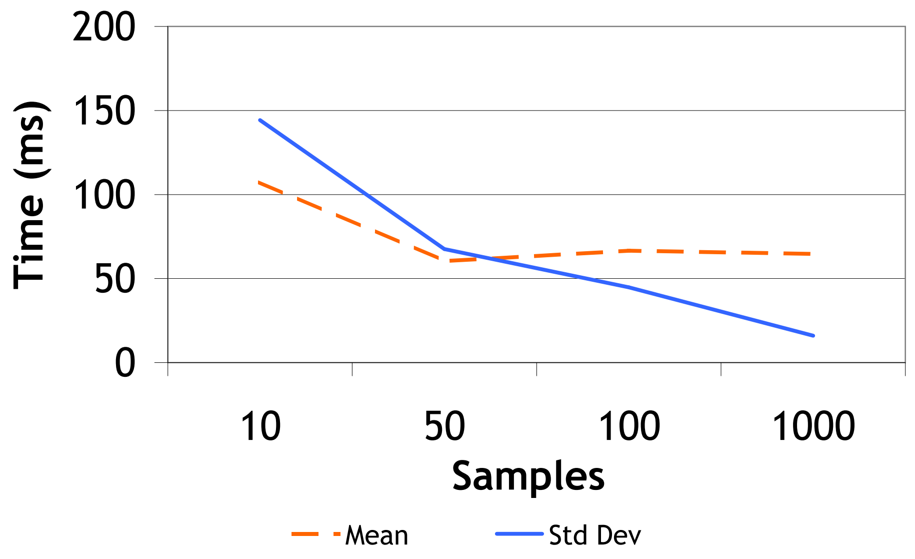


Figure 5.2: Effect on standard deviation as number of samples increases

that there is indeed some warming-up happening in the JVM. The standard deviation is shown by the solid line and decreases with the number of iterations. This means that as the number of times the code is executed increases, its performance becomes more steady and predictable. Unless otherwise stated, the performance results presented in this chapter are based on the recording of 1,000 samples using the benchmark harness. This results in mean values with a very low standard deviation, giving confidence that the results are accurate representations of the true performance.

5.1.2 Data Set

The spatial data set used throughout this chapter is $3Km^2$ of 1:10,000 vector mapping data of Dublin city from Ordnance Survey Ireland (OSI). This data includes building outlines, green areas, street names, rivers and roads. This data is augmented with base map data of the worlds coastlines and the Irish road network. This data set was copied to the mobile device, stored locally and added as a data source to load on initialisation in the middleware's configuration file (Appendix A.1). The details of the test data set are listed in Table 5.1.

5.1.3 Results

Performance evaluations are conducted on the HP iPAQ mobile platform (Table 5.3 on page 157) using the instrumentation tool presented above. Where more than one

Shapefile	Geometry	Description	Size
area_-outlines.shp	Polygon	Area and building outlines for $3Km^2$ OSI data set augmented with data extracted from Trinity College CAD drawings.	14 KB
annotation.shp	Point	String-based descriptive map labels (96).	2.75 KB
roads.shp	Line	Road and path centers from $3Km^2$ OSI data set and for wider Dublin city from Open Street Map[4].	768 KB
world.shp	Polygon	Free base map data for all countries in the world showing coastlines and political boundaries from ESRI.	4.05MB

Table 5.1: Test spatial data set

implementation exists for a pluggable component, both are evaluated and compared. The following framework components are evaluated:

1. The responsiveness of map-based interfaces developed using the frameworks map rendering service.
2. The MRDB in terms of its spatial data retrieval performance and memory usage.
3. The generalisation process, including the generalisation steps: elimination, simplification and clipping.
4. Coordinate transformation.
5. Visibility determination.

5.1.3.1 Map-based Interface Responsiveness

Figure 5.3 shows the time taken to update the interface as a result of a pan (Appendix A.4, Figure A.3), zoom (Appendix A.4, Figure A.1), or rotate (Appendix A.4, Figure A.2) event. The values for each of the events were randomised for each execution of the benchmark, that is, a different pan amount, zoom factor and rotation angle was used for each iteration. Updating the interface requires the execution of the querying algorithm (Section 4.1.4), coordinate transformation (Section 3.3.4) and rendering (Section 3.3.2) for each layer. As such, this benchmark is representative of the overall performance of the core middleware services of the framework.

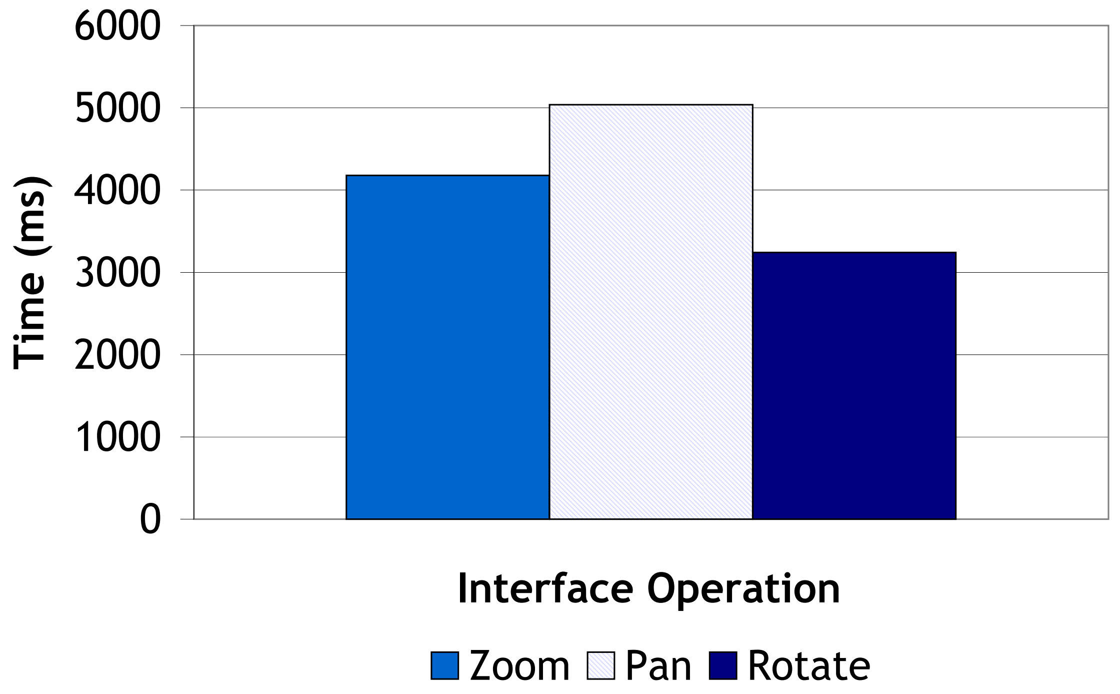


Figure 5.3: Map interface responsiveness

The chart (Figure 5.3) shows that the response time is between three and five seconds, which is well within the tolerance for an acceptable delay (0-12 seconds)[5]. The MRDB, model-oriented generalisation and coordinate transformation algorithms contribute to maintaining a dynamic model of the user's environment and providing spatial services based on this model. To analyse how each of these operations contribute to the overall responsiveness, they are each benchmarked individually.

5.1.3.2 Multiple Representation Database

The MRDB makes spatial data available in different resolutions and levels of detail to allow for flexible zooming on small displays. Two benchmarks have been created to illustrate the performance of the MRDB. The first benchmark compares the speed in retrieving geometry from the extended multi-scale data format with the speed of retrieving geometry from a standard Shapefile. The second illustrates the memory usage of MRDB.

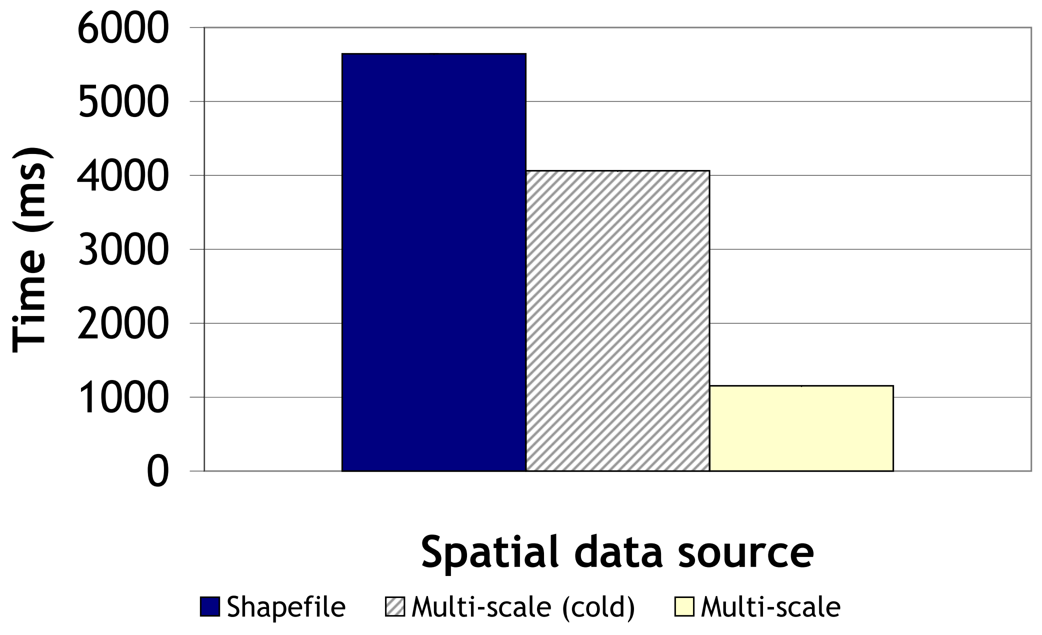


Figure 5.4: Spatial data format performance

Spatial Data Retrieval

Figure 5.4 shows the performance of the retrieval of geometry for a particular area from a standard Shapefile, versus the performance of the retrieval of geometry for the same area from the MRDB. In all cases, the spatial query resulted in the retrieval of 3,683 points requiring the reading of 133.24 KB (the cold multi-scale benchmark also wrote 75.6 KB to memory). The query extents were $200m^2$, representative of a typical area that would be visible on the map-based interface of a spatially-aware application. The MRDB is evaluated in two states: 1) a cold state with no pre-computed LoDs cached; 2) a warm state where geometry at an appropriate resolution has already been computed and cached. The cold multi-scale data structure performs slightly better than a standard Shapefile despite this step requiring the generalisation of geometry, creation of a new LoD and the writing of geometry to the corresponding files. This process happens the first time geometry is retrieved from the Shapefile at a particular resolution, after which a warm data structure is available. The warm data structure is capable of significantly out performing the standard Shapefile because the geometry is less complex, requiring less I/O.

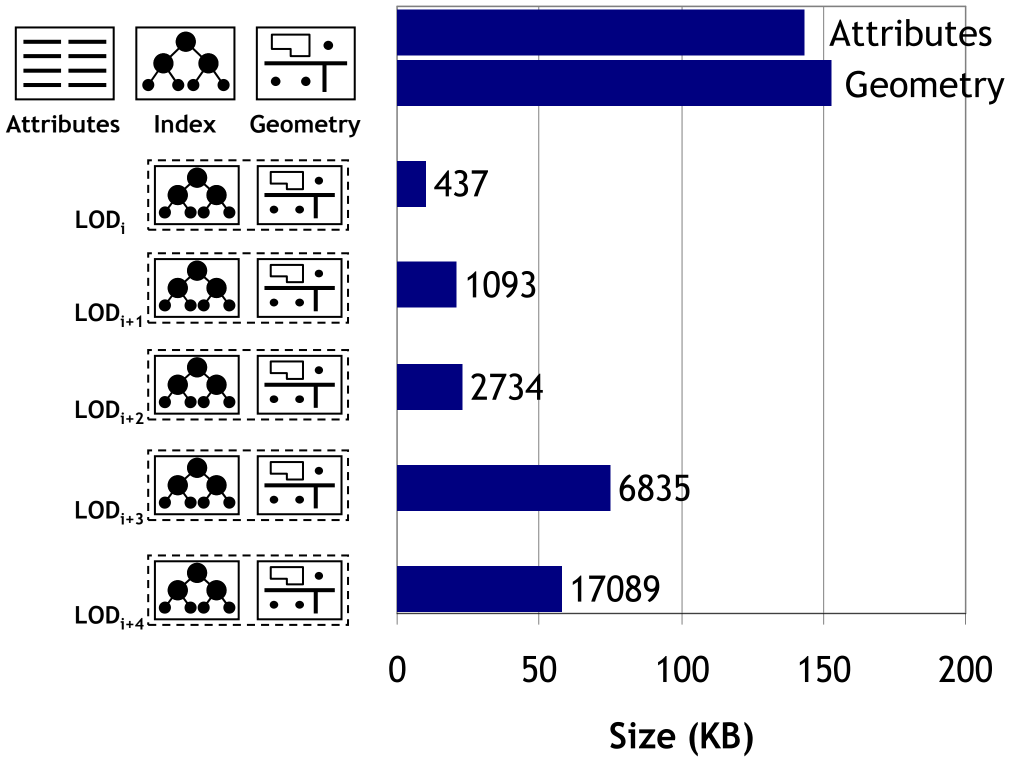


Figure 5.5: MRDB memory usage

Persistent Memory Usage

The MRDB uses caching to store redundant copies of spatial objects at multiple LoDs. This approach trades computation for memory usage, improving performance. To understand the extent to which additional memory is consumed over and above a standard Shapefile, the size of the LoD files belonging to a warm data structure are measured.

Figure 5.5 illustrates the memory usage of various parts of the multi-scale data structure. On the Y axis the multi-scale files for each scale level are listed. At the top of the chart the original geometry and attribute files of the main files are shown to occupy 150 KB and 142 KB respectively. The remainder of the bars on the chart represent the memory consumed by each level in a fully warmed-up multi-scale data structure. The overall increase in memory usage is less than 25%, where memory usage is calculated by summing the size of each of the multi-scale files. This additional memory overhead consumes limited memory resources of the mobile device, limiting the volume of spatial data that can be cached. However, this cost is a trade-off with the components performance benefits (Figure 5.4).

[5]Sub-second updating of the interface is possible when a smaller area or less detail is visible.

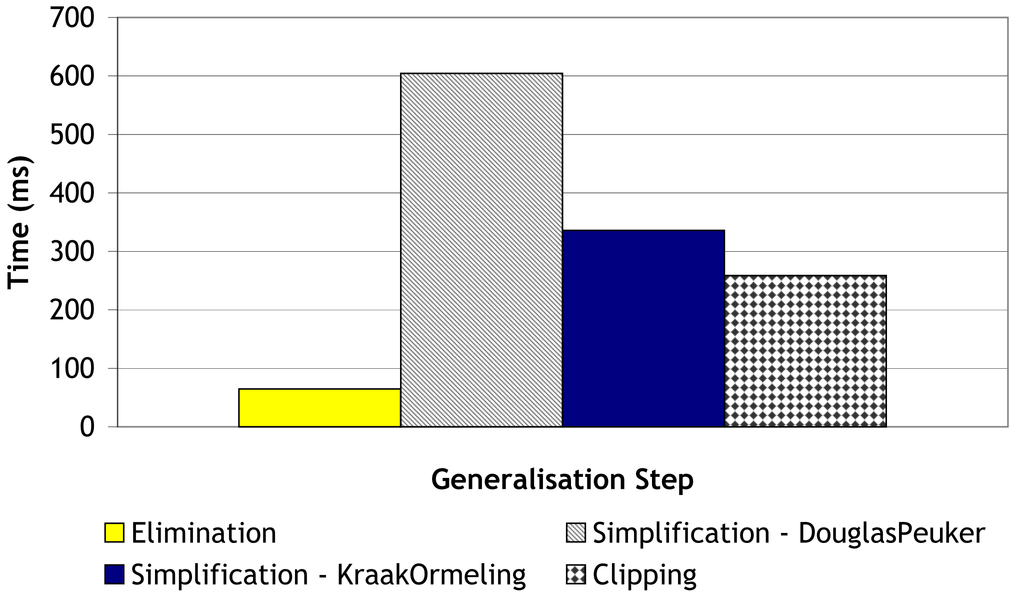


Figure 5.6: Relative performance of generalisation steps

5.1.3.3 Generalisation

The generalisation process is a combination of elimination, simplification and clipping (Figure 5.6). The elimination generalisation step is a simple filtering process and is the quickest of the generalisation processing steps. On each iteration of the elimination benchmark, a scale between 1:400 and 1:25000 is used. In experiments, the elimination processing step resulted in an average reduction in complexity[6] of 3.8%. The simplification processing step is illustrated by two bars in the chart. These bars represent the performance of Douglas-Peucker and Kraak-Ormelling implementations of the pluggable simplification interface. These simplification algorithms succeeded in reducing the complexity of the geometry by a further 58% or 18% respectively. Clipping is the final component of the generalisation process. The clipping algorithms evaluated were Cohen-Sutherland (line clipping) and Sutherland-Hodgman (polygon clipping). The clipping process was applied to both building outlines and coast outlines resulting in a reduction in complexity of 75%[7]. This reduction in complexity results in reduced processing time and power consumed by accessing the spatial data at multiple levels of detail on hand-held devices (RQ-2).

[6]Complexity is quantified as the number of points (or coordinate pairs) required to represent a spatial object or set of spatial objects within a spatial extent.

[7]In this case the entire coast of a country was a single polygon with a small area visible on the device.

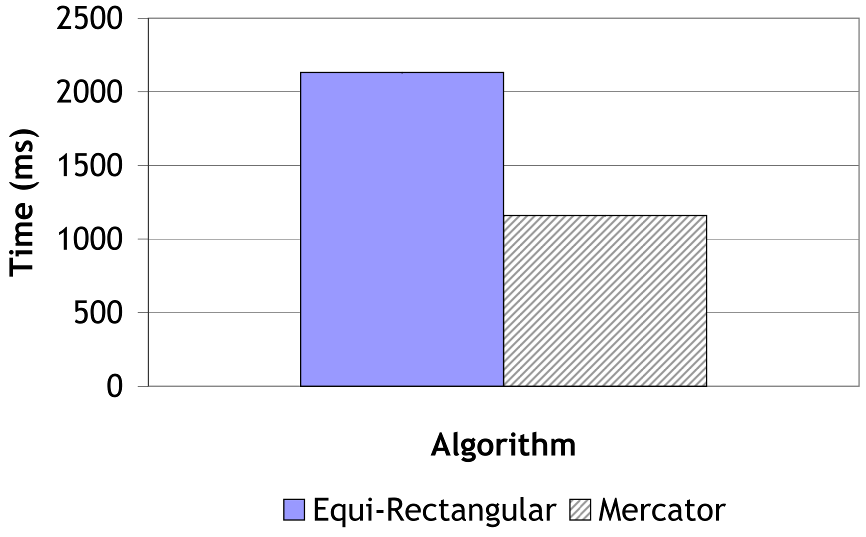


Figure 5.7: Map projection algorithm performance

5.1.3.4 Coordinate Transformation

The final component that contributes to enabling local spatial services on mobile devices is coordinate transformation. A coordinate transformation service renders map-based interfaces from vector spatial data and can be processor intensive. Figure 5.7 compares the performance of two map projections: Mercator and Equi-rectangular. The map projections are extended to incorporate a rotation transformation to support track-up map-based interfaces that orientate themselves in the user's direction of travel. In the benchmark experiments a total of 8,856 points were projected at a random scale and angle of rotation. This number of points is representative of the number of points a location-aware application would typically require to render an accurate map-based interface on a mobile device. The graph shows that the map projection alone consumes roughly two seconds, a significant proportion of the 12 second budget for a responsive application.

5.1.3.5 Visibility Determination

Visibility determination is an innovative spatial service that determines the visibility of real world objects from the user's point of view (PoV) using the multi-scale 2D environment model. Figure 5.8 illustrates the capabilities of this service through an example. The bottom left contains the output from visibility determination. Above is a panoramic photograph of the corresponding real-world area, highlighting the buildings

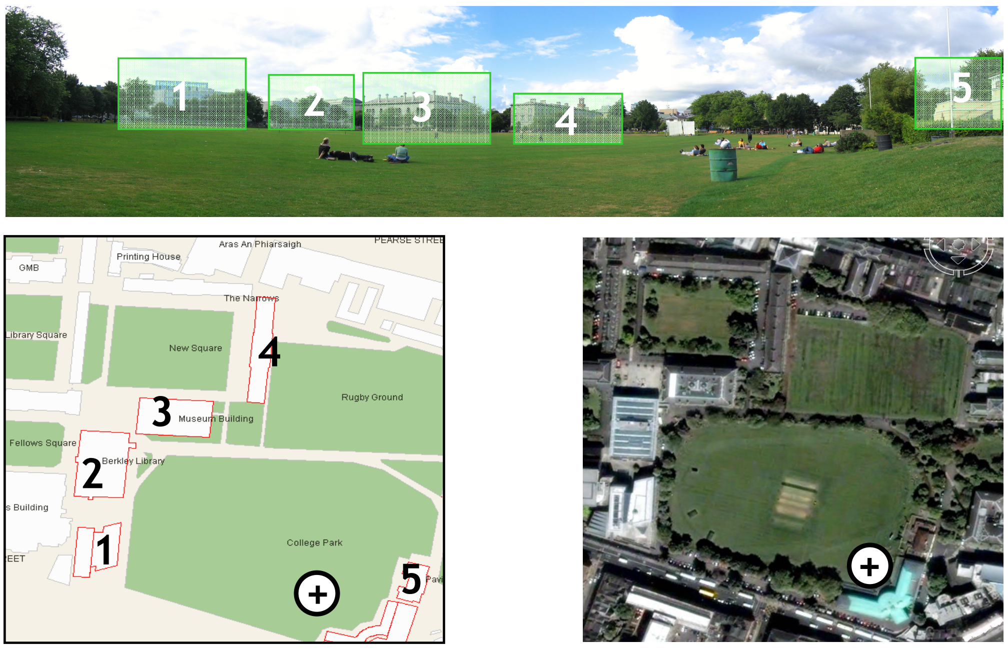


Figure 5.8: Visibility determination example

that the visibility determination service has identified as visible. On the lower right, a top down ariel photograph of the same area is shown. This example is taken from an actual execution of the visibility determination service, illustrating its accuracy. Although the most obviously visible objects have been found, there are objects in the background visible in the photograph that have not been found. This is due to the objects being partially occluded. To improve the efficiency of the visibility algorithm, it does not determine degrees of visibility and classifies partially occluded objects as not visible. Although this approach would not be suitable for rendering a scene in computer graphics, it is adequate for use by mobile devices to infer additional useful context about the user's environment. In addition, the fact that there is no spatial data for the trees in the map means that their effect on visibility was not taken into account. Had they existed in the map, they would have been treated as additional occluding objects, increasing the amount of time taken for the algorithm to complete.

Figure 5.9 shows the performance results for execution of the visibility determination service on the mobile device. There are three different algorithms compared. The first is backface occlusion culling, the second ray casting and finally the execution of both algorithms consecutively on the same depth buffer. The figures for this experiment include the time taken to perform view frustum culling (Section 3.3.6.1), depth buffer sorting (Section 3.3.6.2) and the occlusion culling algorithm (Section 3.3.6.3).

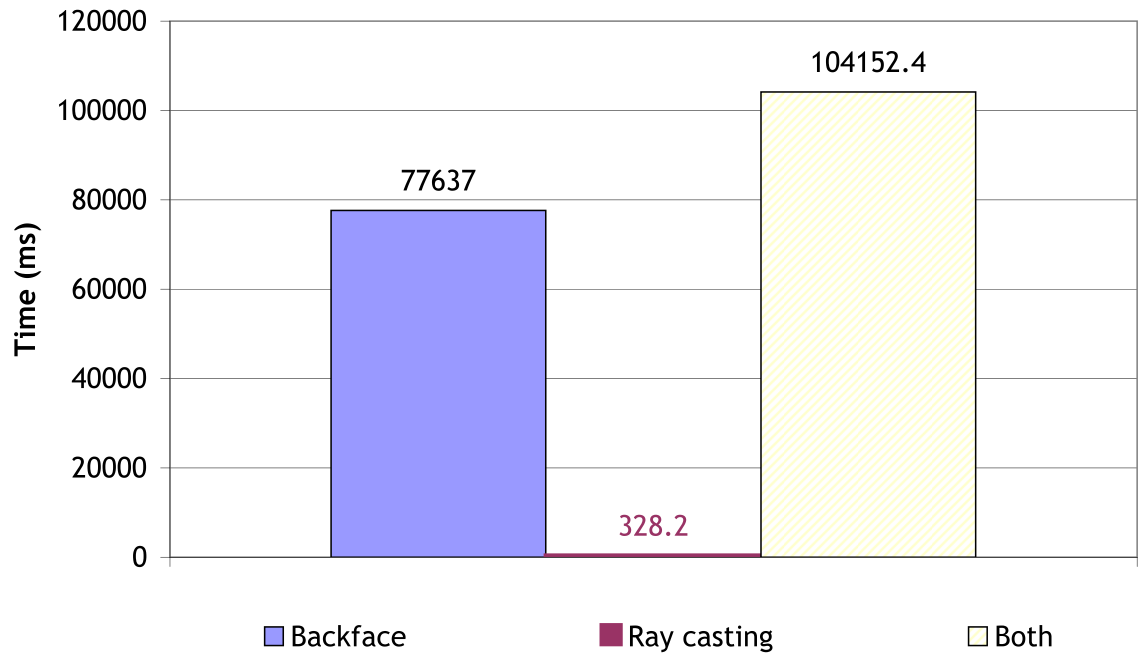


Figure 5.9: Visibility algorithm performance

Each of the benchmarks were executed on 275 unsorted spatial objects retrieved from the test data set (Table 5.1 on page 145), of which 65 were visible. The benchmarks were repeated ten times for each algorithm because of the length of time each sample took.

The backface occlusion culling took just under 80 seconds, ray casting occlusion culling took a little more than 0.3 of a second and executing both algorithms together to improve accuracy took 104 seconds[8]. The performance of these algorithms is due to the lack of hardware support for certain mathematical functions on the mobile platform. To illustrate this, the same benchmark was run on a fully featured JVM on a desktop computer. In that environment, 400 visible objects could be identified from 15,000 in less than 0.2 of a second using backface occlusion culling and 0.01 of a second using ray casting.

[8]The algorithms were executed serially, with backface culling performed first and its output being input to the ray casting algorithm. Their total execution time is greater than the sum of the two algorithms individually because the design of the `VisibilityCulling` interface requires the constructor of realising algorithms to take a `DepthBuffer` as a parameter. This requires the benchmark harness to instantiate a new instance of both algorithms within the timed loop.

5.1.4 Analysis

Overall, the performance of the framework, illustrated by the map-based interface responsiveness (Figure 5.3), is well within the 12 second response time threshold. This allows us to conclude that the framework is capable of providing common application-level spatial services to spatially-aware applications without overburdening the limited resources of hand-held devices and without relying on wireless network connectivity. This success can largely be attributed to the design of the MRDB which provides efficient access to multi-scale spatial data by balancing the need to minimise computation with the use of memory.

The comparison of two pluggable implementations of the simplification generalisation step illustrate a trade-off exists between performance (in this case quantified by reduction in data volume) and fidelity [75]. The Kraak-Ormelling implementation was found to be almost twice as fast as Douglas-Peucker but at the cost of increased distortion and less overall reduction in data volume (Figure 5.6).

The coordinate transformation performance evaluation found the adapted mercator projection to be more efficient than the naïve equi-rectangular projection. This demonstrates the value of applying algorithms and techniques from the field of GIS to mobile application development and argues for an increasingly inter-disciplinary approach to the development of spatially-aware applications.

The implementation of the visibility determination service has been shown to be accurate, but hardware and JVM limitations precluded us from achieving the full benefits of the service. However, the service is the first middleware-based example of such a service that is capable of executing on a mobile device. The service illustrates the potential of the middleware approach for developing novel spatially-aware application services. Its failure to meet performance requirements does not negate the utility of the rest of the framework.

Having demonstrated that it is now possible to locate spatial data on mobile devices and provide spatial operations and application-level services without relying on unreliable wireless networks to access centralised servers, the question is, which approach is better? To help answer that question, the next section presents an evaluation of both these approaches from an energy consumption point of view.

	Advantage	Disadvantage
Local	No communication cost Lower latency	Application less responsive Energy consumed by processor
Remote	Less energy consumed by processor Application more responsive	Energy consumed by communication Security and privacy issues Network reliance Variable responsiveness due to network latency

Table 5.2: Local versus remote execution of tasks

5.2 Energy Consumption

Mobile devices rely on a finite energy source that is a major constraint in mobile environments [72]. To maximise the amount of work the user can do before the battery becomes discharged, hardware and software designers have to optimise energy consumption to avoid either heavier battery packs or short durations between battery charges [41].

Hardware designers have tackled this challenge by turning off idle components [128] and reducing the voltage level of the processor [237]. Software designers, however, have focused on computation offloading (remote execution) [7]. Computation offloading is a trade-off between two of the major components of power consumption in mobile devices: communication and computation (Table 5.2). To offload a task, the mobile client sends the parameters of the task to a server; the server executes the task and sends back the results. Energy is saved if the energy consumed sending the parameters and receiving the results is less than the energy that would be consumed by executing the task locally [41]. A likely candidate task for offloading will have a small transmission size and a long computation time [137].

However, there are a number of challenges associated with offloading tasks to servers. Firstly, if the goal is to conserve energy it must be possible to estimate the amount of energy required to: a) perform the task locally; and b) communicate the task to a server, wait for the task to complete and then receive the results. Making these estimates is challenging as they are effected by factors such as the transmission bandwidth, power consumption, and the network congestion [137]. It is also necessary to take the device responsiveness into account. Servers are more powerful than mobile devices and can typically perform any task in a fraction of the time that the mobile device would take. However, communication of the task and results between mobile client and server is a source of latency. There are also many tasks that can not be offloaded because they require access to data on the device, are tightly integrated with other local components or would require the transmission of privacy sensitive information.

There has been a lot of work in investigating the interaction between mobile devices and servers. Othman and Hailes [173] performed simulations to show that battery life can be extended by up to 21 percent by offloading tasks. Anand et al. analysed the power consumption of location-aware applications in the SmartCampus service-oriented middleware, concluding that energy can be conserved by minimising the communication between server and client [11]. Flinn et al., through their experiments with PowerScope, discovered that changing application fidelity[9] can lead to significant energy savings by reducing the volume of data to be transmitted over the network [75]. This finding is supported by Siewiorek et al. who estimates that nearly 80% of the power consumed by mobile devices can be due to communications [211]. The work by Siewiorek et al., on wearable computers further concludes that trading off energy expensive communication for local computation can result in significant energy savings. Li et al. [128] developed a partitioning scheme that minimised the energy consumed by programs running on hand-held computing devices which are connected to a server. It was found that of 18 different programs, 13 consume less energy when they are offloaded. The remaining applications consumed less energy when they ran locally, illustrating that for different classes of application, different offloading strategies are appropriate[10] and for some applications it is more efficient to perform all computation locally. However, none of these energy evaluations considered the unreliability and frequent disconnections that are experienced by hand-held devices due to mobility. These factors further motivate the migration of computation to the mobile device whenever possible.

The model for spatial middleware presented in this thesis locates all spatial data and

[9]Fidelity is an application-specific metric of quality. For example, the amount of geographic features included on a map would be an appropriate measure of fidelity for an application with a map-based interface.

[10]Applications where no energy benefit to offloading was found included speech and image compression / decompression and decryption. This was largely due to the high communication volume required to offload the computation.

spatial middleware services locally on the device, thereby eliminating the energy that would otherwise be consumed by interaction between servers and clients. This maps to an assumption of a computing environment, where users will not always be connected. The research question is whether placing all the computation on the mobile device, in order to minimise dependence on communication, reduces or increases energy usage for the spatially-aware class of mobile applications (RQ-4). This section presents a comparative energy evaluation of two possible approaches to providing spatial services. In one approach, the framework is used to provide all spatial services locally and in the second approach, a server provides the same service. By analysing the relative execution time and energy cost of computation and communication for each approach we show that it is more energy efficient for spatially-aware mobile applications to trade expensive wireless communication for local processing.

5.2.1 Methodology

An experiment is conducted to measure the system-level power consumption of the same spatially-aware application developed using two design approaches:

1. The framework is used to provide all spatial services locally.

2. The mobile devices accesses server-based spatial services over a wireless network.

The hand-held device used for the evaluation was the HP iPAQ hx2490 Pocket PC [104]. The characteristics of this devices are listed in Table 5.3. The server was a Dell Latitude D400 laptop with Intel Pentium M 1.3 GHz, 1GB RAM and built in 802.11b wireless networking. The mobile devices and server communicated over a 802.11b network interface in ad-hoc mode. These were the only two devices on the network, minimising network latency as contention is not an issue and congestion is not possible on a single-hop network. The laptop, taking the role of the server, was dedicated to serving the single client.

Figure 5.10 illustrates the basic approach behind the `EnergyEvaluationUserBot` class that replays simulated user interactions with a spatially-aware application. A spatially-aware application has been implemented that displays a map with buildings, green areas, roads and labels on the devices screen (Figure 5.11). The `EnergyEvalu-ationUserBot` class is implemented as a thread, that once started will call `waitFor-`

CPU	520 MHz Intel PXA270 Processor
Memory	256 MB total memory (192 MB ROM and 64 MB SDRAM) Up to 128 MB user available persistent storage memory
Battery	Rechargeable 1440 mAh Lithium-lon
Wireless Network	WiFi (802.11b) Bluetooth wireless technology Serial IR

Table 5.3: HP iPAQ hx2490 Pocket PC specifications (Source: [104])

`ClientIdle()`, which causes the thread to wait until all threads in the framework have finished (threads are used to retrieve spatial data from the MRDB and render it to map layers). The user simulator will then trigger a user interface event that causes the application to behave exactly as if a user has just zoomed or panned the map. This event causes map projection to be updated, additional geometry to be read at the specified scale for individual layers and a redrawing of the interface[11]. When this process is complete the user simulation will continue by firing the next event. The sequence of events and their parameters are hard-coded to allow the same sequence to be played back on different versions of the application.

The second design approach requires that the application access spatial services (in this case map rendering) over a wireless network. To achieve this, the framework was extended with a new map layer, `RasterTileMapLayer`, and a minimal map tile server was written (Figure 5.12).

The `RasterTileMapLayer` class extends the abstract `DynamicMapLayer` class. Instead of retrieving geometry for producing a map-based interface using the frameworks `QueryEngine`, the `RasterTileMapLayer` has a socket connection to the server. Map tiles of 100px square are passed from server to client over this connection (in the same way as web map interfaces [91, 241, 156, 145, 239]). On the server-side, a simple application opens a `ServerSocket` and executes an instance of the middleware framework. The server's job is to accept incoming socket connections, read requests for map tiles, use the map rendering service of the framework (Section 3.12) to render the required tile, compress then serialise the image [41] and transmit it to the client. In order to make the comparison as fair as possible, common performance optimisations were implemented on the server including: thread pooling and reuse to decrease latency,

[11]The sequence of events raised by the `EnergyEvaluationUserBot` is shown in Appendix A, Listing A.3.

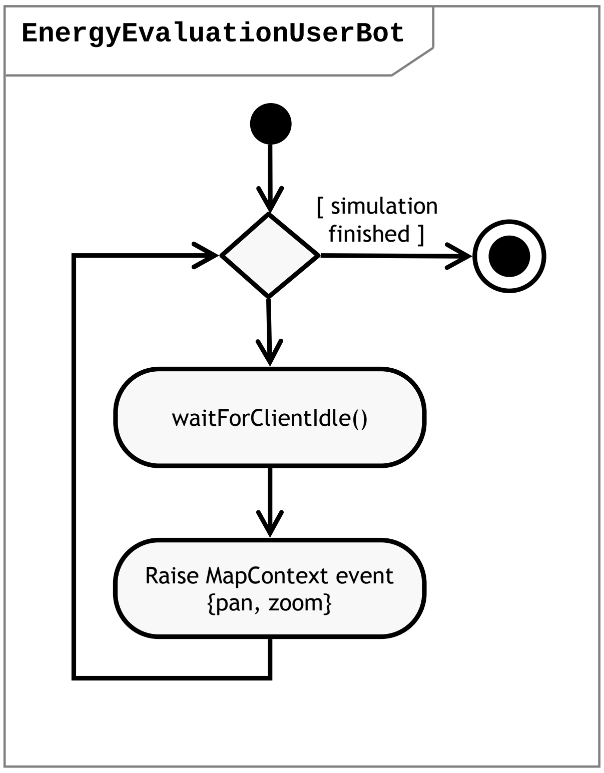


Figure 5.10: Energy evaluation user bot

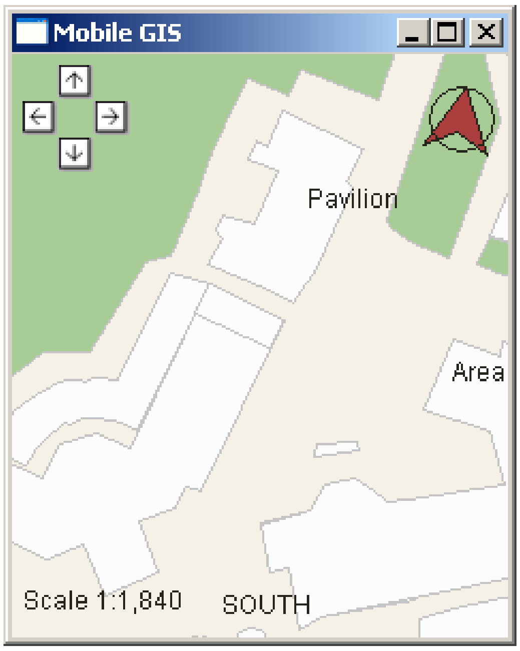


Figure 5.11: Spatially-aware application map interface

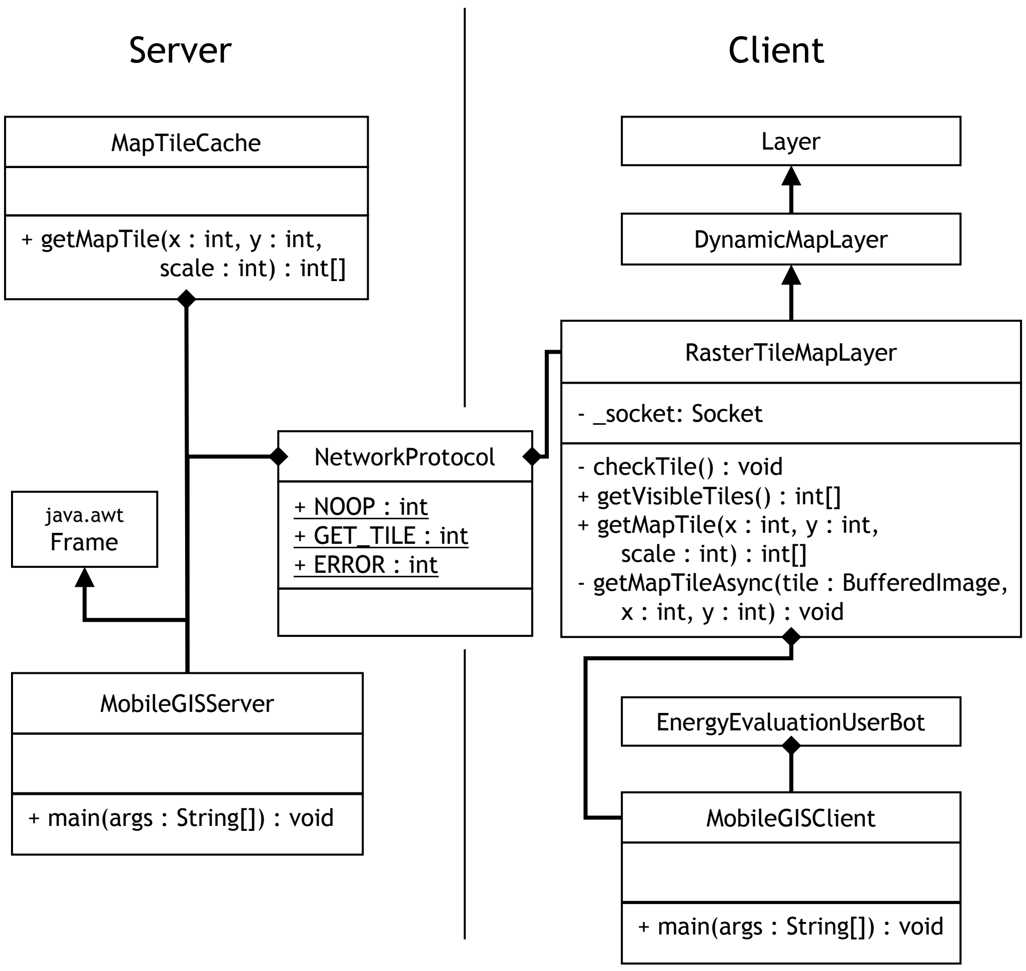

Server
Client
MapTileCache
+ getMapTile(x : int, y : int, scale : int) : int[]
java.awt Frame
NetworkProtocol
+ NOOP : int
+ GET_TILE : int
+ ERROR : int
MobileGISServer
+ main(args : String[]) : void
Layer
DynamicMapLayer
RasterTileMapLayer
- _socket: Socket
- checkTile() : void
+ getVisibleTiles() : int[]
+ getMapTile(x : int, y : int, scale : int) : int[]
- getMapTileAsync(tile : BufferedImage, x : int, y : int) : void
EnergyEvaluationUserBot
MobileGISClient
+ main(args : String[]) : void

Figure 5.12: Framework extensions for server-based spatial services

caching of pre-rendered map tiles as images on the servers file system and the compression of map tiles to minimise network latency. A `NetworkProtocol` class defines three types of static message types that can be passed between the server and client:

- `NOOP` Short for no-operation. This indicates an empty message and is sent from the client to the server to hold a socket connection open or to check the server is still responding.
- `GET_TILE` This indicates that the message is a 'get map tile' request. This message is sent from client to server and contains parameters specifying the top left coordinate of the map tile required and its scale.
- `ERROR` This message indicates that the server could not complete the request. Common causes of this would be is the server did not have any spatial data for an area that a tile was requested for or if a map tile was requested at a scale that was not available.

The experimental setup is illustrated as a sequence diagram in Figure 5.13. Above the dashed line, the energy evaluation of the version of the application using the framework with all computation performed locally is shown. In this case, the firing of an event starts the timer, t_1. The event causes the map layer to retrieve spatial data from the query engine and render it. Once complete, control returns to the user simulator, at which point t_1 stops and the simulator moves on to the next event in the set.Below the dashed line a version of the application is executed that accesses server-based spatial services over a wireless network. The `EnergyEvaluationUserBot` fires the same events, recording the time using timer t_2 but the `RasterTileMapLayer` retrieves the interface from the server instead of rendering it from spatial data.

Energy consumption data was gathered using, *acbTaskMan 1.4cf*, a software tool that provides a power meter and can log battery power levels in the background while other applications run (Figure 5.14). This same method of gathering data was used by Anand et al. [11]. An alternative, and arguably more accurate, approach to measuring energy usage would be exchange the devices battery itself for hardware that both supplies power and measures current usage [96, 75]. The software approach was chosen as it makes use of built in instrumentation in the hand-held device and allows for the

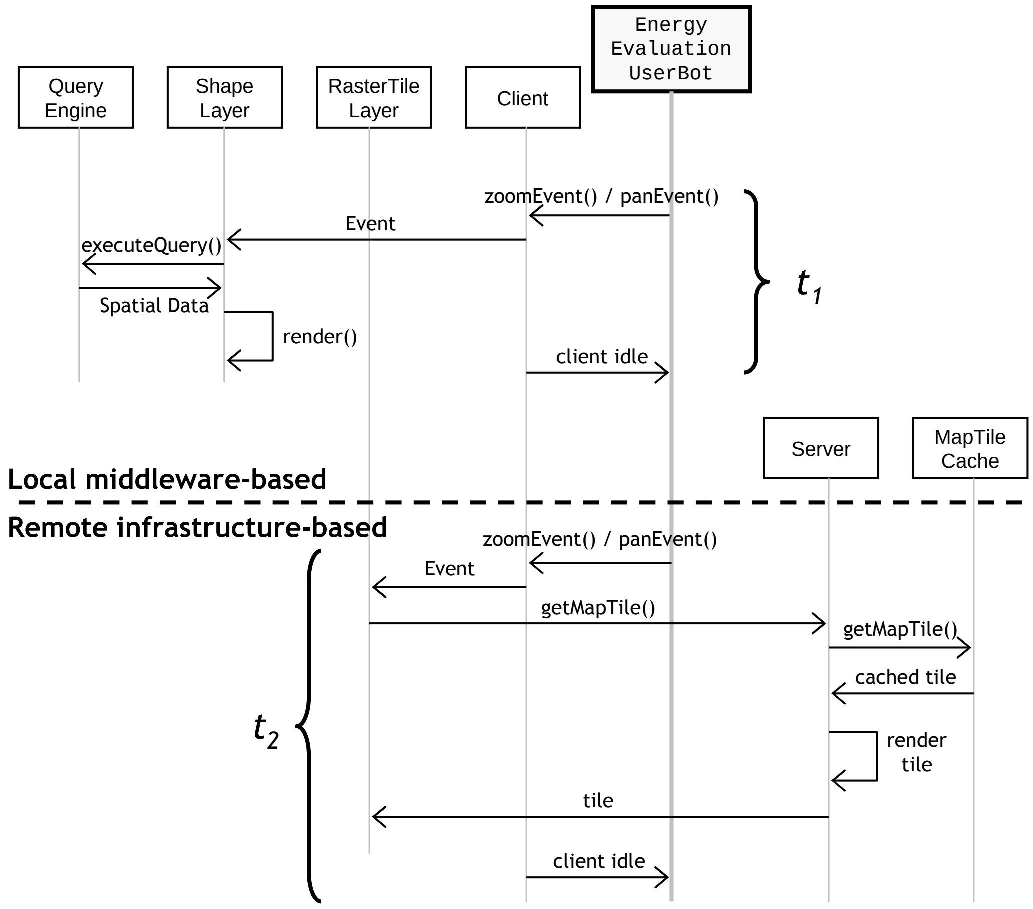


Figure 5.13: Energy consumption experimental setup

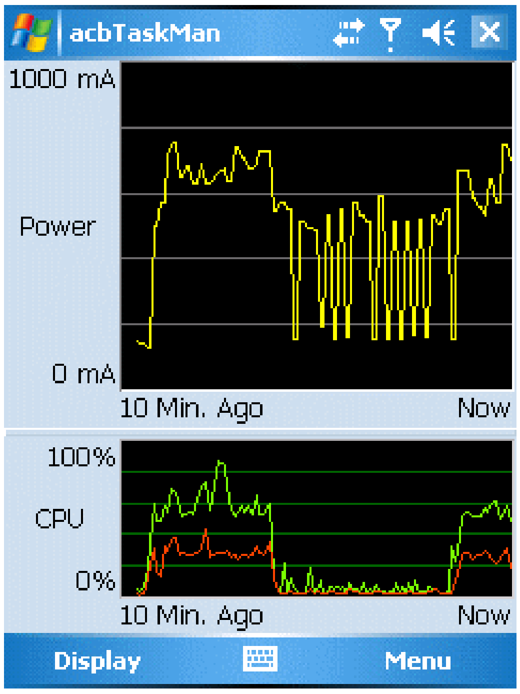


Figure 5.14: acbTaskMan PDA power meter (Source: [3])

collection of a number of additional parameters such as network activity that help us place the energy readings in context.

Throughout the experiment, the screen saver and screen back-light remained off. The MRDB extended Shapefile format had its geometry caches for each level of detail emptied before each run of the application, requiring execution of the generalisation operations to produce geometry at multiple scales. This was done for fairness, ensuring the middleware-based approach does not already have all the necessary geometry generalised and cached at the necessary levels of detail. The total elapsed time for the sequence of user interactions for each version of the program ($\sum_{1}^{60} t_1$ and $\sum_{1}^{60} t_2$, where 60 is the number of events that the `EnergyEvaluationUserBot` fires) and the electrical current drawn on the hand-held device were measured. Where CPU usage figures are shown, these are isolated to the spatial application process and do not take into account any background processes running on the device.

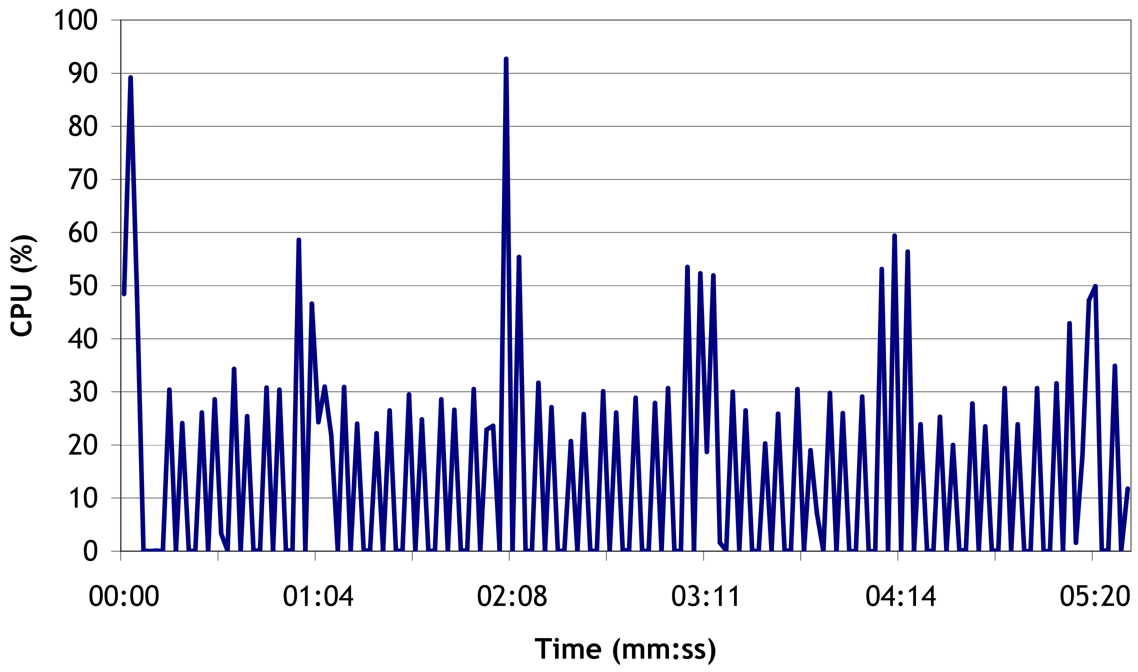


Figure 5.15: CPU usage over time (middleware)

5.2.2 Results

This section presents the results of measuring energy consumption for each of these approaches to providing spatial services to hand-held devices. Energy consumption is quantified by the simple equation:

$$energy = current * time \tag{5.1}$$

The current is measured as the average current drawn per second during execution of the application and its unit is the amp (A). The current drawn is measured using the milliampere-hour (abbreviated as mAh) unit. Energy is measured in coulombs (symbol, C). One milliampere-hour is equal to 3.6 coulombs, where 1Ah is the electric charge transferred by a current of one ampere for one hour. Time is measured in seconds.

5.2.2.1 Local Computation

To put the energy consumption in context, the processor usage, memory usage and network activity were also measured. Figure 5.15 shows the CPU usage by the spatially-aware application for the duration of the experiment. It can be seen that it took five minutes, thirty seconds for the application to complete the set of 12 actions five times

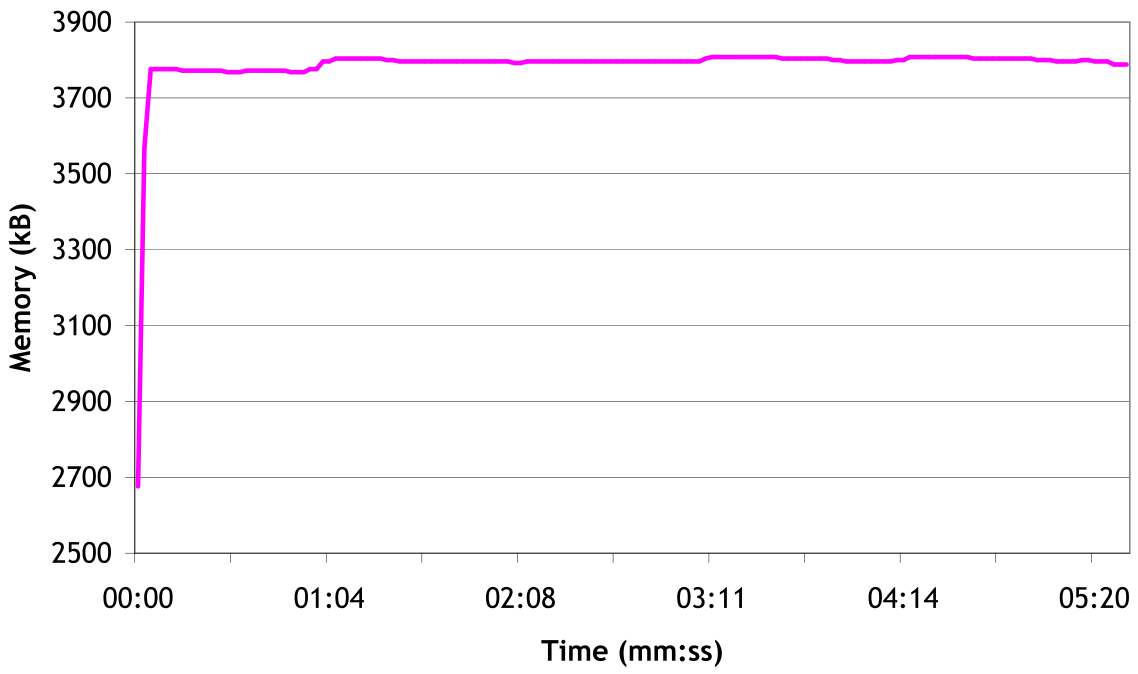


Figure 5.16: Memory usage over time (middleware)

on the mobile device (Listing A.3). The spike in processor usage at the end of each of the five repetitions of the set of events corresponds to zoom events which require accessing spatial information at a different level of detail. The variability in CPU usage (illustrated in the graph by the CPU usage regularly dropping to zero) is a feature of the `EnergyEvaluationUserBot` which waits for the application to idle, takes a pause and then triggers the next event.

The memory usage of the program is illustrated in Figure 5.16 and the network activity is shown in Figure 5.17. The memory usage grows quickly as the application is launched and then remains relatively steady for the duration of the execution. Slight changes in memory usage are a feature of the JVM garbage collector rather than any explicit freeing of memory resources within the application. The network activity shows only eight UDP packets. This is because the wireless card was powered on (in power save mode) but the application did not need to use the network. The NIC was left on during the experiments for fairness. It allows the results of the two approaches to be compared based on their software architectures rather than on hardware configuration. Having the NIC on is also more representative of the application's behaviour in the real world as the NIC would have to be powered to enable the peer-to-peer collaboration facilities of the Hermes framework to acquire spatial information.

The energy consumption for the execution is illustrated in Figure 5.18. The energy

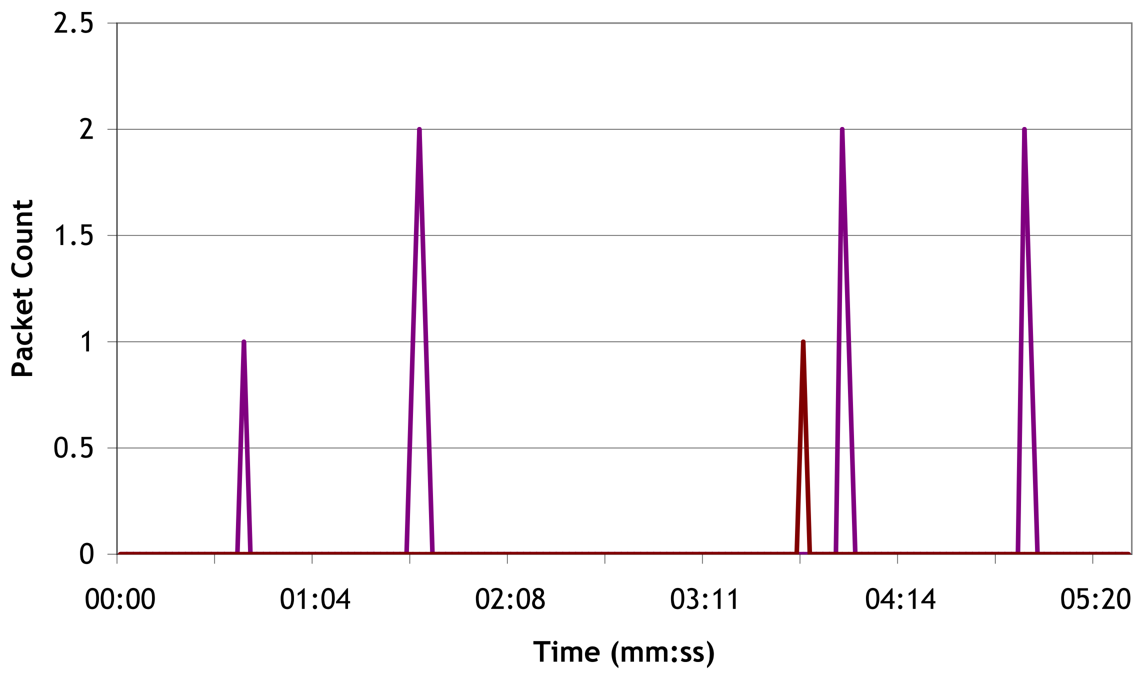


Figure 5.17: Network activity over time (middleware)

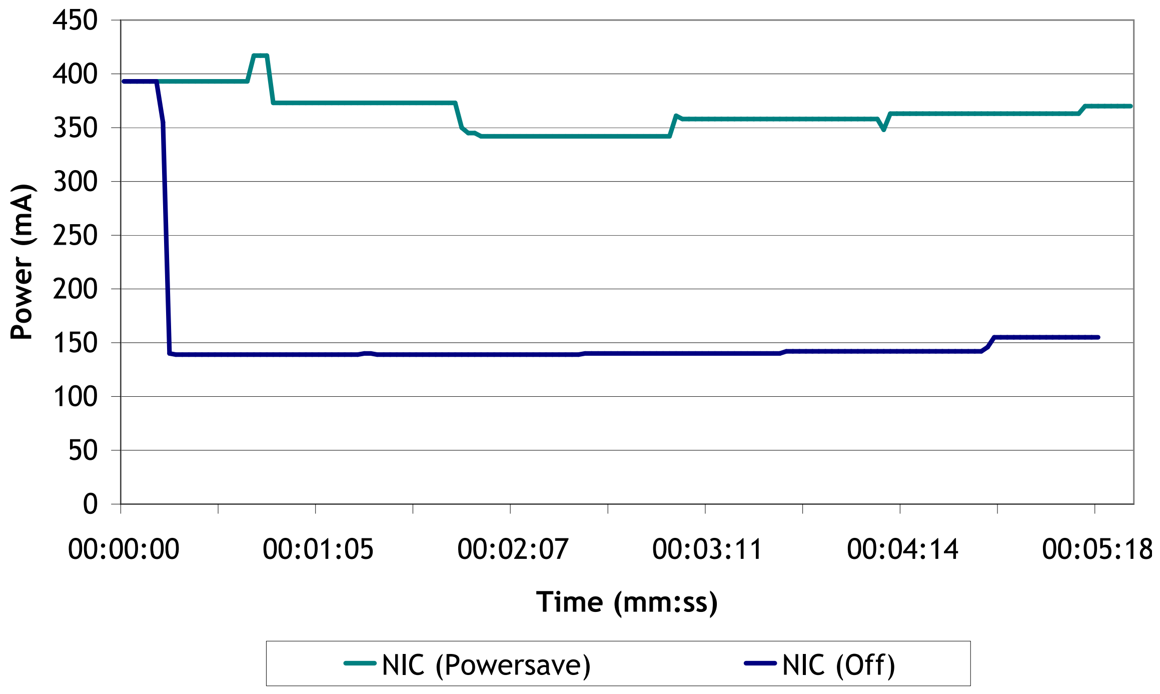


Figure 5.18: Power drain over time (middleware)

consumption for the case where the NIC was turned on (but in power save) versus the case where the NIC is turned off is shown to highlight the significant effect the wireless network hardware has on energy consumption, even when the network is not in use. We analyse the power consumption based on the data collected when the NIC was powered on.

The total set of events took 5 minutes 30 seconds to execute ($\sum_{1}^{60} t_1 = 330$ seconds). The average energy consumption for that period was 364.65 mA. Since we know that a charge of 1mA transferred for one hour (3,600 seconds) is 3.6C we can calculate the total energy consumed using Equation 5.1 as:

$$330s * 0.36465A = 120.3345C$$

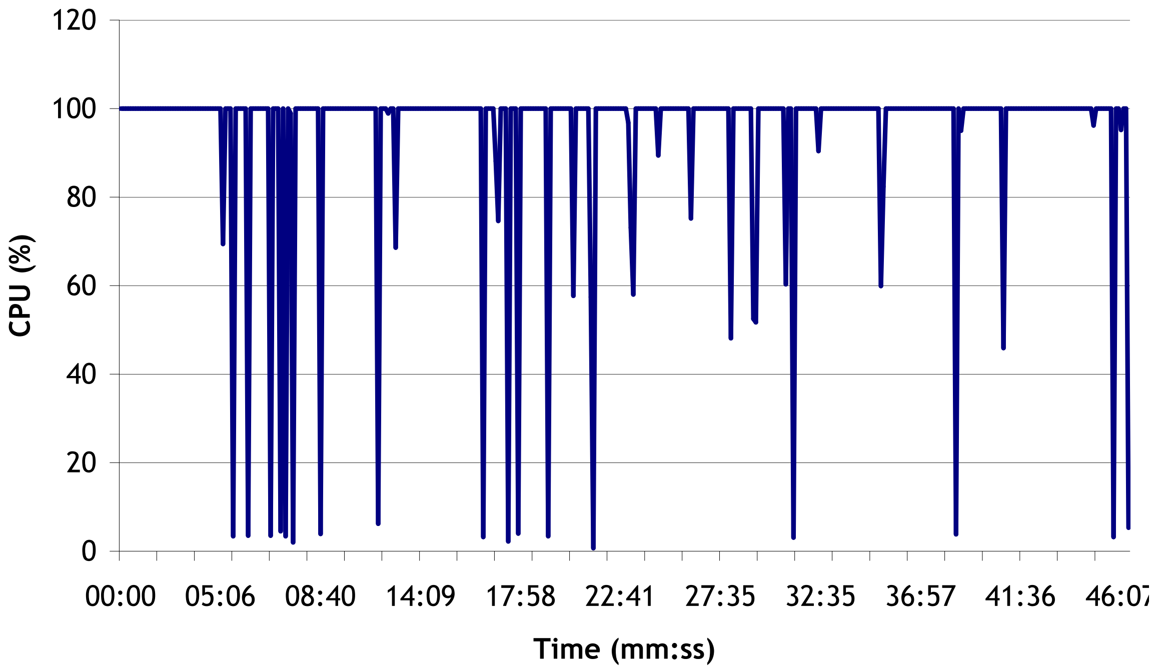


Figure 5.19: CPU activity over time (server-based)

5.2.2.2 Server-based Computation

The second approach to providing spatial services to mobile spatially-aware applications is via server-based services. This section presents the results of performing the same set of user actions but this time the application produces a map-based interface by requesting raster map tiles from a server rather than rendering the interface from spatial data. As in the evaluation of the network independent version of the application, the processor activity, memory usage and network activity are first presented to put the total energy consumption in context. Figure 5.19 illustrates the CPU usage during the execution of the application. The first contrast to note with the corresponding graph (Figure 5.15) is the length of time that the execution took. In this case it took 46 minutes, 21 seconds to perform the same set of operations ($\sum_{1}^{60} t_2 = 2781$seconds) and the average CPU usage for that period was 94.41%. The periods where the application idled between events remain visible.

Memory usage in this case was higher, more variable and grew as the application executed (Figure 5.20). This is due to memory consumed by caching raster map tiles. The network in this case remained active for the entire execution of the application with TCP packets passing in both directions (Figure 5.21).

Due to the higher processor usage and constant network activity, this version of a

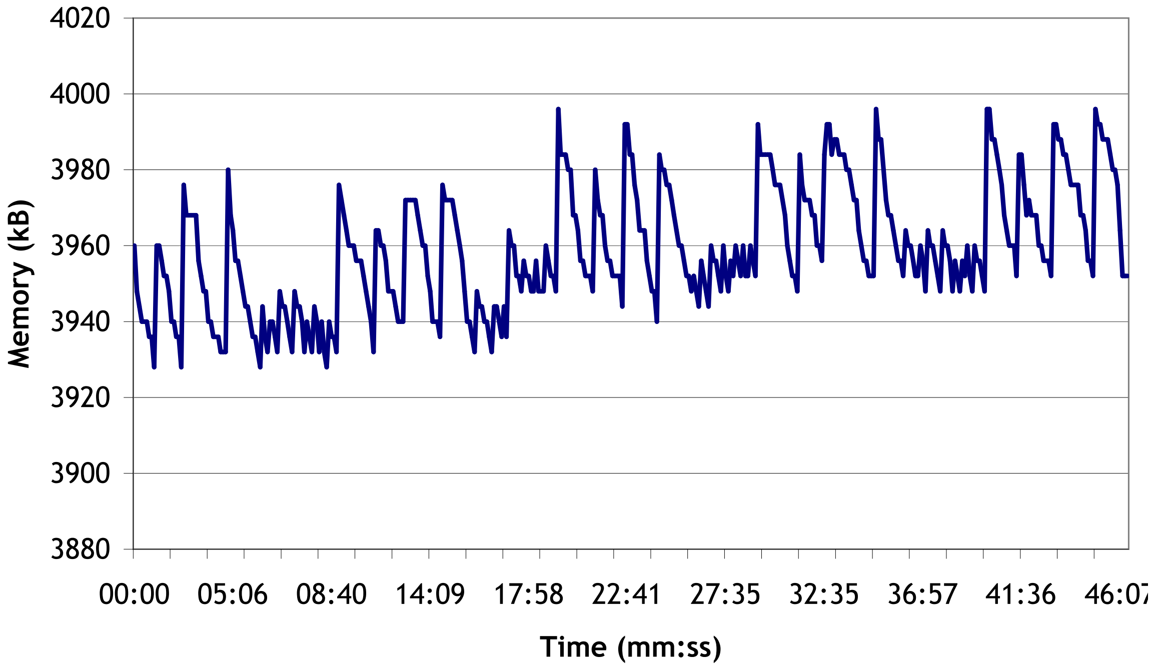


Figure 5.20: Memory usage over time (server-based)

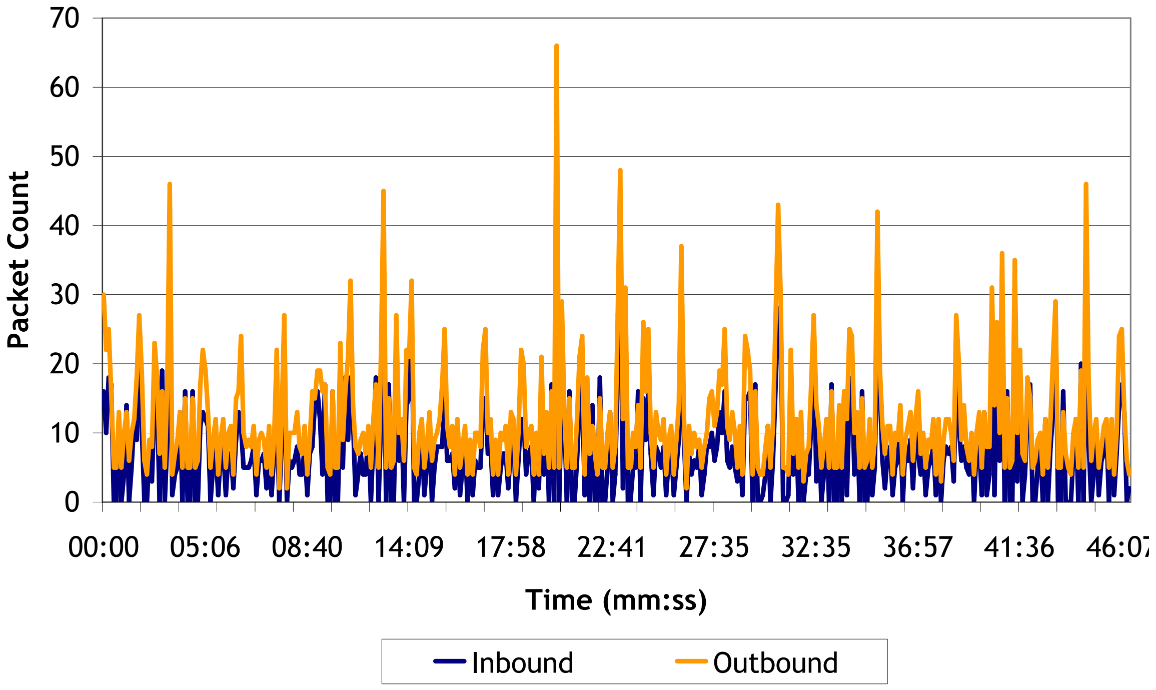


Figure 5.21: Network activity over time (server-based)

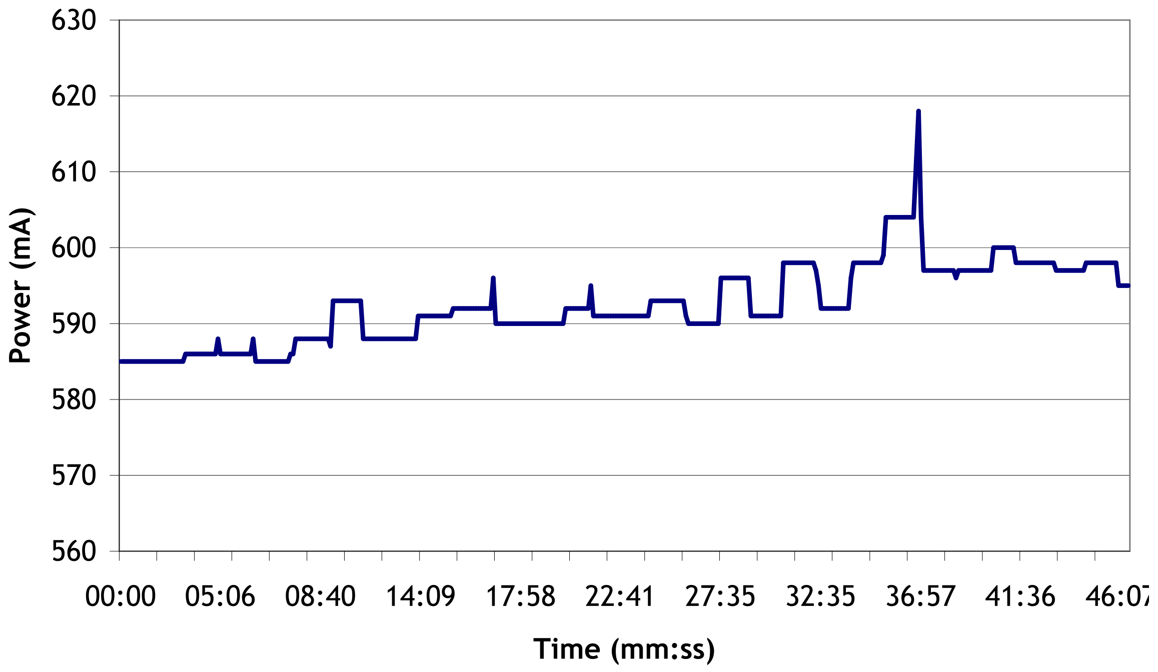


Figure 5.22: Power consumption over time (server-based)

spatially-aware application consumed much more power (Figure 5.22). In this case the average energy consumption was 592.43 mA and the set of events took 2781 seconds to execute. We calculate the total energy consumed using Equation 5.1 as:

$$2781s * 0.59243A = 1647.56C$$

5.2.3 Analysis

The use of server-based spatial services consumed $1,527.23C$ more energy than providing the same spatial service from middleware on the mobile device. Figure 5.23 illustrates the energy usage of both approaches on one graph, highlighting the difference in the length of time each application took. This difference can be largely attributed to network latency. For every event, the server-based version of the application had to retrieve an average of six 100px square map tiles varying between 20kB and 70kB (depending on results of compression). In addition, the server-side version required the processor on the mobile device to perform the de-compression and conversion from a collection of bytes to an image for display on the device screen. This turned out to be very processor intensive, largely because it was done without any hardware assistance and suffered from poorly performing image manipulation mechanisms in the

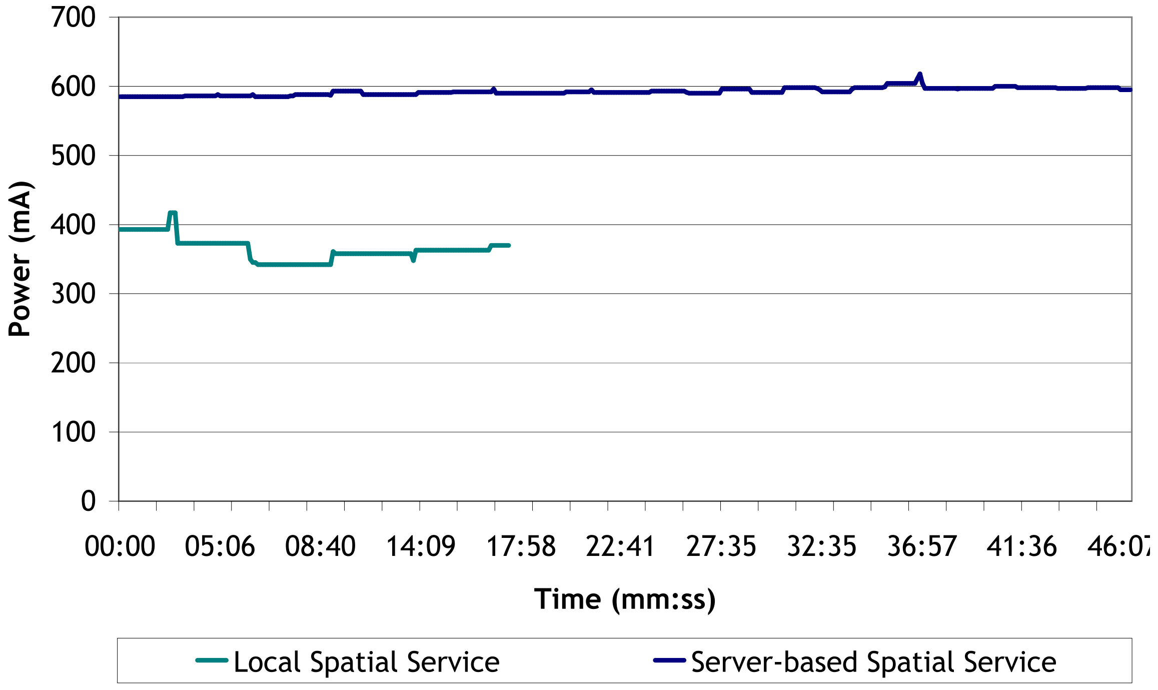


Figure 5.23: Comparison of power consumption

JVM. Figure 5.23 also shows that even if using the server-base spatial service resulted in the same application performance and experiment execution time, it would still have consumed more energy.

To help illustrate the difference in energy consumption between these two approaches, the energy consumption can be expressed based on battery life. The battery in the mobile device used for these experiments has a charge capacity of 1440mAh (5184C) (Table 5.3). In the case of providing the spatial services as middleware on the device, the battery would be depleted in 3 hours, 56 minutes, 56 second. Using the server-based spatial service this battery would be depleted in only 2 hours 25 minutes, 50 seconds[12] (38% less battery life).

Based on these results, the model for mobile spatial services presented in this thesis consumes less energy and less computational resources than existing approaches for spatially-aware applications that support rendering and interacting with map-based interfaces.

It should be noted that the figures for the power consumption of the middleware based approach (Figure 5.23) assume that the spatial data exists locally on the device. This thesis assumes a collaborative computing environment, where mobile devices dis-

[12]Assuming no other applications or services are running on the device and an ideal battery model with a constant capacity for all discharge profiles [187].

cover and serendipitously communicate with peers to acquire and disseminate spatial data. In such an environment, if a device enters an area for which it has not spatial data cached, it will attempt to acquire the data from other devices or infrastructure in the environment. This process will consume energy at the same rate as the server-based approach, but only for a short period of time, after which, the energy consumption will return to the lower level. Quantifying the energy consumption due to collaboration in peer-to-peer overlays in mobile ad-hoc networks is beyond the scope of this work but is discussed as possible future work in Section 6.2.1.

5.3 Framework Reusability & Extensibility

The generic framework for spatially-aware mobile applications (Section 3.4) addresses the common challenges posed by hand-held mobile devices in a reusable and extensible manner. The framework consists of a collection of classes, both concrete and abstract, that can be reused, extended, and composed to reduce the cost and improve the quality of applications [73]. To be considered useful, the application framework must be capable of serving as the basis to a range of applications and support the addition of application-specific behaviour. Therefore, framework components are required to be both reusable and extensible.

To illustrate and evaluate the framework's ability to be extended and reused, a case study application has been developed that uses the framework as its basis. The use of a case study application is a form of qualitative rather than quantitative evaluation that is an accepted form of framework evaluation [32]. Case studies can be used to illustrate the capabilities of a framework by evaluating the degree of reuse and the ease of extensibility afforded by the framework in the development of a case study application.

This section presents the design and implementation of a single case study application, *Where Are We ?*[13]. This spatially-aware application locates mobile users and displays their locations on an adaptable map-based interface. Users' positions are continually updated and the display scaled and panned to keep the user in the centre and the other mobile users visible in the map extents.

5.3.1 Case Study Application

Figure 5.24 illustrates the extensions and reuse of the framework to support the *Where Are We ?* application. The application consists of two application specific classes, `WhereAreWe` and a `ContextMapLayer` (highlighted in the diagram with a yellow background). The `WhereAreWe` class is the main class of the application and is responsible for initialising the Hermes framework components that it will require. It does this by first creating a new `ContextSource` corresponding to a GPS sensor. The framework's acquisition component is then instructed to acquire positioning context from this sen-

[13]Flyvbjerg has illustrated that it is possible to generalise on the basis of a single case [76].

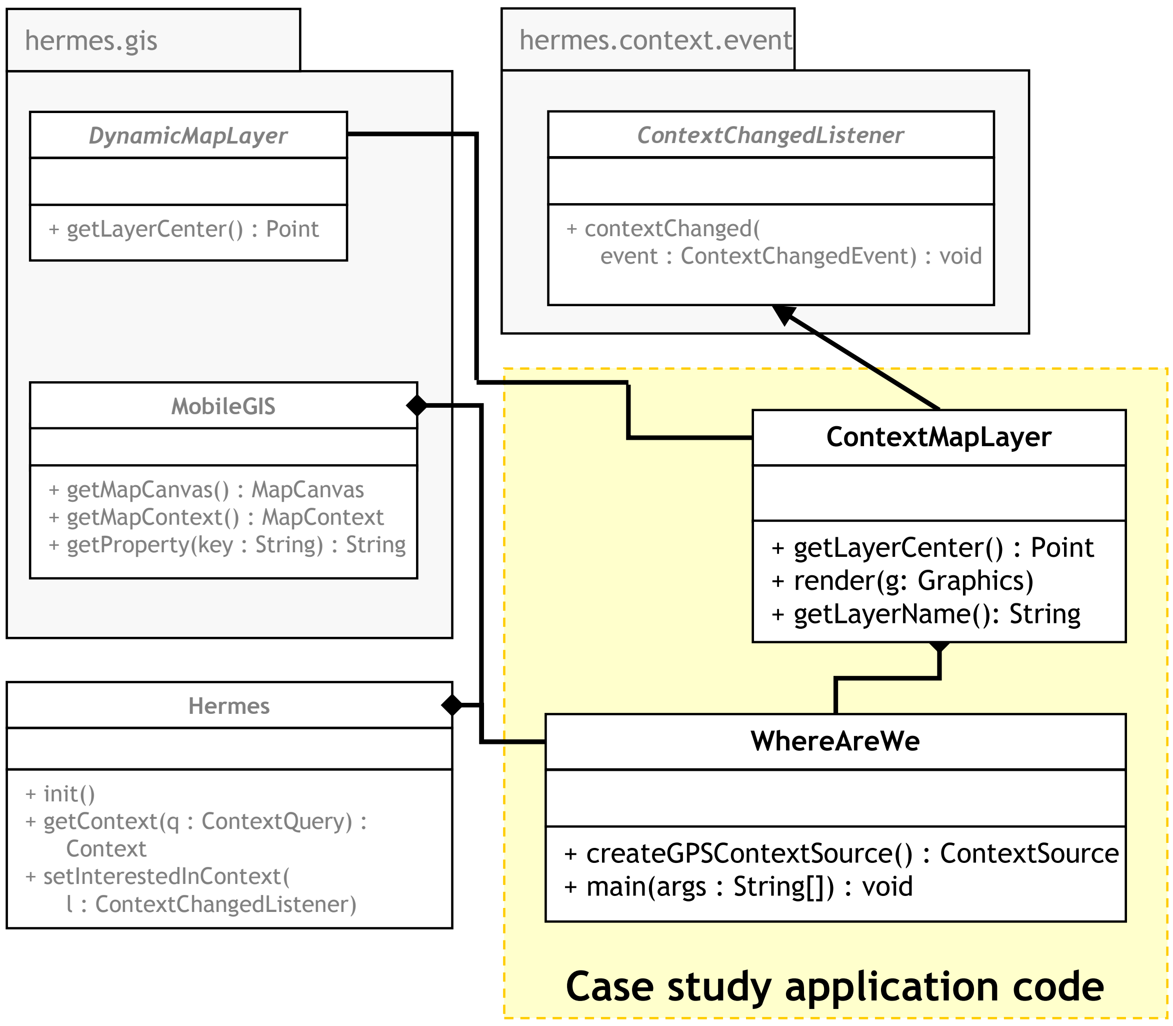

hermes.gis
DynamicMapLayer
+ getLayerCenter() : Point
MobileGIS
+ getMapCanvas() : MapCanvas
+ getMapContext() : MapContext
+ getProperty(key : String) : String
hermes.context.event
ContextChangedListener
+ contextChanged(
event : ContextChangedEvent) : void
ContextMapLayer
+ getLayerCenter() : Point
+ render(g: Graphics)
+ getLayerName(): String
Hermes
+ init()
+ getContext(q : ContextQuery) :
Context
+ setInterestedInContext(
l : ContextChangedListener)
WhereAreWe
+ createGPSContextSource() : ContextSource
+ main(args : String[]) : void
Case study application code

Figure 5.24: Case study application implementation

sor. An instance of the `ContextMapLayer` is created and Hermes is informed that this new class is to be informed of any changes to position context. The `ContextMapLayer` is an extension of the abstract `DynamicMapLayer` class, implementing the `render()` method. The `render()` method is responsible for drawing mobile users' locations on the map using the frameworks coordinate transformation service to convert from latitude and longitude to screen coordinates. The `MapContext` class is then instructed to produce an interface using a specified style and showing specific map features. The `ContextMapLayer` is added to the interface. The Hermes framework now begins acquiring positioning information from both the specified GPS context source and peer devices. This context is delivered to the *Where Are We ?* application, producing the interface illustrated in Figure 5.25.

The steps to develop a spatially-aware application using framework can be generalised from this case study as:

1. Specify pluggable component implementations, initially available data sources, default scales and location in the configuration file (Section 4.3.2 on page 134).
2. Write a main class that constructs a user interface and calls `Hermes.init()` and `MobileGIS.init()`.
3. Set up the context sources in Hermes (Section 4.0.3 on page 109).
4. If the application interface incorporates a map, use the `MapCanvas` component.
5. Use the spatial query model to define the spatial data that should be included in each map layer and add the layers to the map (Section 4.1.4 on page 118).
6. Extend `MapLayer` to define a custom style to render layers or to add functionality (Section 3.3.2 on page 81).
7. Call methods included in the geometry model and query engine to add additional spatial reasoning to the application (Sections 3.3.3 and 4.1.4).

5.3.2 Analysis

Where Are We ? contains code that both extends and directly reuses functionality provided by the framework. The level of reuse is measured to assess whether the

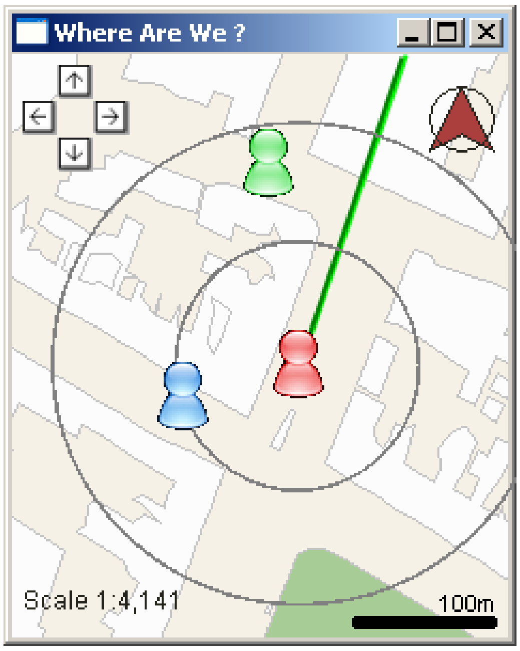


Figure 5.25: *Where Are We ?* case study application interface

framework achieves its goal of reducing the cost of development of spatially-aware applications by providing a generic model for spatial middleware services (RQ-5). Reuse level is the standard metric for measuring the amount of software reuse in an application [180] and is generally expressed as a percentage of the total source lines of the application. The case study application consists of two classes, `WhereAreWe` and `ContextMapLayer`. In total, the application consists of 149 lines of Non-Commented Source Statements (NCSS). The framework consists of 10,882 NCSS (8,960 excluding benchmarking and debugging tools) giving a code reuse of 99.07%. It should be noted that although the application is developed on top of the framework, it does not make use of all its functionality. The framework loads components at run-time making it difficult to get an accurate measure of the quantity of framework functionality used by the application. For this reason the 99.07% code reuse is in relation to the total number of NCSS in the framework and not the total number of NCSS executed by the case study application.

5.4 Evaluation Summary

This chapter has described the evaluation of various aspects of the framework for spatial middleware services. The evaluation has shown that the algorithms for accessing spatial information at multiple levels of detail are capable of supporting responsive spatially-aware applications on hand-held mobile devices. The overall performance of the frameworks key algorithms and middleware services are summarised in the map-based interface responsiveness evaluation, which found the delay in updating the interface to be well within the user acceptable 0-12 second range. This achievement is contributed to by many of the individual components of the framework. The MRDB provides efficient access to multi-scale geometry, performing significantly faster than a standard Shapefile but consuming 25% more memory. The model-oriented generalisation operations successfully reduce the complexity of spatial data, highlighting the contribution of minimising geometry complexity through on-demand generalisation as a key means of improving performance of spatially-aware applications. Coordinate transformation supporting rotation was shown to consume roughly two seconds, which is a significant proportion of the 12 second budget. The implementation of the visibility determination service demonstrated, for the first time, that is it possible to perform accurate visibility determination in middleware on a mobile device. However, hardware and JVM limitations limited performance of the service.

The approach to maintaining a dynamic model of the user's environment locally has been shown to be significantly more energy efficient than the more common approach of off-loading complex spatial operations to a server. The use of the framework to store and manipulate spatial information and generate a vector-based map interface dynamically, consumed significantly less energy that accessing a server-based service even when the network hardware is powered on. This illustrates the disproportionate amount of energy consumed by the wireless network interface in relation to other sources of energy consumption such as the screen and processor. The battery life of spatially-aware mobile applications can be maximised by minimising the use of the network interface, further motivating the need for algorithms such as the ones presented in this thesis to provide spatial services without depending on network connectivity. Although the model for spatial middleware does not rely on constant network connectivity it does assume that it is possible to periodically connect to peer devices and infrastructural components for the purpose of acquiring spatial data.

A case study spatially-aware application illustrated the reusability and extensibility of the generic framework. These two properties ensure the framework can support the development of a range of mobile spatially-aware applications.

The following chapter concludes with a summary of the most significant contributions of this thesis, commenting on the overall benefit of the approach and discusses research issues that remain open for future work.

Chapter 6

Conclusions and Future Work

This thesis describes the design and implementation of a model for spatial middleware featuring a locally maintained environment model and common spatially-aware application services. The model for spatial middleware features algorithms designed to minimise the processing time and power consumed on hand-held mobile devices while providing uninterrupted access to spatial data at multiple levels of detail. The environment model and spatial middleware services are incorporated into a generic framework that supports the development of spatially-aware applications by contributing implementations of spatial services that addresses the common challenges posed by hand-held mobile devices in a reusable and extensible manner. This chapter summarises the achievements of the work, outlines its contributions to the state of the art and concludes with a discussion of research issues that remain open for future work.

6.1 Achievements

This thesis presentes a middleware approach to maintaining a model of the user's environment where spatial data is disseminated using the collaboration services of the Hermes framework. Having acquired spatial data, a multiple representation database makes spatial data available in different resolutions and levels of detail. The implementation of the middleware approach manages complex spatial data in real-time on commercially available hand-held devices. The design demonstrates that infrastructure-based spatial services are not always necessary and that the increasing capabilities of mobile devices are making it possible for application developers to design increasingly complex applications for mobile devices.

While the approach of maintaining a local environment model on mobile devices mitigated many of the challenges of providing reliable access to spatial information in a mobile environment, it also posed the challenge of implementing lightweight versions of computationally intensive spatial services. This thesis described the design and implementation of five application-level spatial services. Adaptable map rendering, spatial reasoning, coordinate transformation and route generation represent the most common services required by spatial applications, as illustrated by the review of related work (Chapter 2). These spatial services were evaluated from a performance perspective with the objective to demonstrate that they perform fast enough to meet user performance expectations (identified as a 0-12 second response time). It was found that the implementations were well within this response time, demonstrating that efficient, lightweight versions of GIS services can now be provided in middleware on mobile devices.

A visibility determination service was designed and implemented but hardware and JVM limitations precluded us from achieving the full benefits of the service. However the service is the first middleware-based example of such a service that is capable of executing on a mobile device. As such, the visibility determination service demonstrates the potential of the spatial middleware to support the development of novel interfaces and interaction metaphors.

Chapter 5 explored the energy efficiency of storing spatial data locally versus the more common approach of off-loading complex spatial operations to a server. It found that the use of server-based spatial services consumed 38% less energy than providing the same spatial service from middleware on the mobile device. This highlights the role middleware and application design has in extending the battery life of mobile devices and contributes to the argument that as a general rule of thumb, it is more energy efficient to perform a task on a mobile device if the task can be performed without adversely impacting the responsiveness of the device.

The implementations of the environment model and spatial services were packaged as a generic framework addressing the lack of reusable software for developing spatially-aware mobile applications. Existing frameworks are server-centric or support the development of software in only one application domain requiring developers to repeatedly tackle the challenges of mobility, device resource limitations and unreliable wireless networks. A case study application (Section 5.3.1) illustrated the reusability and extensibility of the generic framework ensuring that the framework can support the development of a range of mobile, spatially-aware applications.

The main contributions of this thesis are summarised as:

- An overview of spatial services for mobile computing, with respect to the architectural models, spatial services, provisions for maintaining availability in the face of unreliable wireless networks, approach to managing the limited resources of mobile devices and the generic support for developing spatially-aware applications. It was found that the majority of existing systems maintain their environment models on servers to which mobile clients rely on wireless networks to maintain access.

- A model for spatial middleware that combines a vector-based model with the ad hoc collaboration features of the Hermes framework to autonomously disseminate spatial data to mobile devices. This model eliminates the dependence on server-based infrastructure for spatial data distribution, simplifying the deployment of spatially-aware applications.

- A local environment model, maintained by a multiple representation database that dynamically generates levels of detail, approximates continuous scale adaption with stepped levels of detail. The local environment model provides both application and mobile middleware developers with efficient access to spatial data at multiple resolutions that can be exploited to provide interactive, engaging interfaces and services to users of mobile devices.

- A set of algorithms for manipulating spatial data to support application-level spatial services. These are: route generation; rendering adaptable map-based interfaces; and spatial reasoning. This set of algorithms preserves limited processing resources, while supporting responsive interactive applications, by selecting data at an appropriate scale, clipping geometry to viewport extents, buffering static objects and caching coordinate transformations.

- An innovative visibility determination service for spatially-aware applications based on the use of a depth buffer, on which a variation of occlusion culling is performed to reduce the set of geometry to a possibly visible set. It was found that the implementation of this service did not meet performance goals on the mobile platform. However, the service illustrates the potential of the middleware approach to developing novel spatially-aware application services and is the first example of such a service that does not take a infrastructure approach.

- A comparison between the model for mobile spatial services presented in this thesis and existing approaches investigates the energy trade-off when rendering and interacting with map-based interfaces to spatially-aware mobile applications. It was found that the model for spatial middleware presented in this thesis consumed significantly less energy than accessing an equivalent server-based service. This discovery will inform the design and development of future spatially-aware mobile applications allowing them to conserve energy and maximise the amount of work the user can do before the battery becomes discharged.

- A generic framework for spatial services, that provides reusable and extensible implementations of spatial services, designed not to overburden the limited resources of hand-held mobile devices and that do not depend on continuous network connectivity. A case study application demonstrated the reusability of the framework and illustrated its extensibility through inheritance and parameterisation of a pluggable component model at specific extension points. This framework lowers the cost of spatially-aware mobile application development by providing implementations of common components and services.

6.2 Future Work

Throughout the process of designing, implementing and evaluating the application framework presented in this thesis, a number of issues worthy of further investigation were identified. This work relates to the measurement of energy consumption of peer-to-peer (P2P) overlays, dynamic factors impacting the decision to offload computation, data structure for flash memory, spatial data integration and the legal and technical restrictions of commercial spatial data.

6.2.1 Energy Consumption of P2P Overlays

This thesis presentes a model for spatial middleware that takes the approach of performing all computation necessary to provide spatial services locally. This model assumes a collaborative computing environment, with mobile devices capable of discovering and serendipitously communicating with peers and fixed infrastructure (Section 3.1.1.1 on page 62). The Hermes framework provides the means to manage the acquisition and dissemination of spatial data (Section 3.1.2 on page 63). As potential users are likely to spend most of their time in familiar environments, the need to acquire new spatial data would not occur often. The performance evaluations of spatial services assumed the most common case where the user is in a location they have been before and have already acquired spatial data describing its geometry (Section 5.1.2 on page 144). As a result of this assumption, the energy consumption incurred during the discovery and acquisition of spatial data was not measured. There has been some work in quantifying the energy consumption due to collaboration and file distribution in MANETs [96, 95]. However, a comprehensive study on the energy consumption in P2P overlays on mobile devices has not been conducted using real devices, running computationally intensive (for example, spatially-aware) applications with realistic workloads. The incorporation of results from such a study is future work for this thesis.

6.2.2 Partitioning of Application Logic

The framework contributed by this thesis gives developers the option of performing tasks previously only possible on a server on a mobile device. This thesis has shown that there are energy savings in providing spatial services from middleware as opposed to as a network service (Section 5.2.3 on page 169). However, these evaluations did not take into account such factors as network congestion and client and server utilisation (Section 5.2.1 on page 156). These dynamic constraints may mean that while performing computation locally may be more energy efficient, there may be situations where, due to other applications on the device, the user would be best served by offloading a task [151]. Conversely, the network conditions and server utilisation at any moment may make a task that is normally more efficiently performed on a server, more energy efficient on the client.

Current energy evaluations and power-aware middlewares do not take into account the dynamic factors that impact the decision to offload computation and are not performed in the presence of multiple concurrent applications [137, 75, 174, 154]. The work in quantifying the energy consumed by the middleware approach taken by this thesis could be further refined by considering these dynamic factors. This knowledge may lead to middleware for mobile devices that can make power-aware decisions on how spatial services are accessed.

6.2.3 Data Structures for Flash Memory

The framework presented in this thesis includes algorithms that allow spatial services to be provided via local computation and local spatial data storage on mobile devices. Mobile devices often feature flash storage devices, which behaves differently to volatile semiconductor memory such as DRAM or non-volatile hard disks which store data on rotating platters with magnetic surfaces [109]. Accessing flash storage is slower than volatile memory (particularly write operations) but can be read and written to faster then non-volatile hard disks (especially when reads and writes are non-contiguous). A bit can only be erased by erasing a whole block of memory, and can only be erased a finite number of times (the memory physically degrades with each write).

The mobile platform, on which the spatial middleware in this thesis was developed, uses flash memory for both program memory and persistent storage (Section 4.0.1 on page 107). The implementation of data structures for multi-scale spatial data and spatial indexes as part of the MRDB revealed that many of the best practices, common optimisations and design patterns for implementing file I/O performed poorly (Section 4.1.3 on page 116). In particular, buffering strategies that read from memory in blocks, reducing the number of read operations, decreased rather then increased performance. Similarly, aligning writes and reading ahead in a stream decreased performance.

The ineffectiveness of these optimisations is because of inherent assumptions about the nature of the underlying memory architecture. A study by Gal and Toledo [86] discovered that data structures optimised for page-based disk access and are not efficient on mobile devices and concluded that flash-aware data structures are needed.

Initial work has been published proposing data structures for specific types of data. For example, Chowdhury et al. present a flash-aware data structure for storing strings [45]. There is a need to further research the requirements for flash-aware data structures. In particular, an efficient data structure for storing spatial data in flash memory is required.

6.2.4 Spatial Data Integration

The model for spatial middleware presented in this thesis assumes that spatial data can be acquired from peer devices through collaborative data dissemination in an ad-hoc networking environment (Section 3.1.1 on page 62). In implementing and evaluating the spatial middleware services, a single data set (Table 5.1 on page 145) was used (partitioned into fragments where necessary). Because these fragments originated from the same data-set, the re-integration of partitioned spatial data was trivial. However, when individual fragments of spatial data are from different data sources (as would likely be the case in a real world deployment), imprecision in the data could cause inconsistencies that would have to be automatically reconciled [153]. The re-integration of possibly overlapping and erroneous spatial data remains a challenge [81, 219]. Further work is necessary to determine how inconsistencies in different spatial data sets can be reconciled without user interaction on a resource limited mobile device.

6.3 Chapter Summary

This chapter summarised the most significant achievements of the work presented in this thesis. In particular, it outlined how this work contributed to the state of the art in mobile computing by providing a generic framework containing lightweight spatial services for developing robust spatially-aware applications that don't rely on a service-oriented infrastructure and maximising the amount of work the user can do between battery charges. The framework has the potential to reduce the cost of spatially-aware application development, while at the same time provides insight into potential approaches and design decisions that can lead to extended battery life on mobile devices. By providing the means to build interactive spatially-aware applications that do not require reliable access to network services, the reliability, availability and usefulness of mobile applications is increased. In turn theses contributions may lead to wider adoption of mobile technology as spatial-awareness leads to more intuitive and context-appropriate application behaviour. The chapter concluded with suggestions for future work arising from the research undertaken in relation to this thesis.

Appendix A

Further Implementation Detail

A.1 Framework Configuration File

Listing A.1: The `gis.properties` configuration file.

```
# Debugging
gis.debug=true

# Networking
gis.net.port=4444
gis.net.host=ancosantoir.dsg.cs.tcd.ie
gis.net.tile.width=100
gis.net.tile.height=100
gis.net.keepalive=30
gis.net.timeout=5

# Screen
gis.screen.width=240
gis.screen.height=300

# Scale
gis.scale=2760.3848
gis.scale.cache.factor=2.5
```

```
# Center of Map
gis.center.lat=53.341442
gis.center.lon=-6.2501383

# Spatial Data Sources
gis.source.count=3
gis.source.buffer=1024
gis.source.multiscale=true
gis.source.index=SSX
gis.source.1.file=data\\area_outlines_WGS84.shp
gis.source.2.file=data\\country.shp
gis.source.3.file=data\\text_WGS84.shp

# Projection
gis.projection.model=EARTH
gis.projection=Mercator

# Generalisation
gis.generalisation.elimination=false
gis.generalisation.elimination.filter=MinimumAreaFilter
gis.generalisation.elimination.tolerance=0.1
gis.generalisation.simplify.tolerance=2
gis.generalisation.simplify.threshold=2000
gis.generalisation.clip=true
gis.generalisation.clip.line=CohenSutherland
gis.generalisation.clip.poly=SutherlandHodgman
gis.generalisation.clip.points.threshold=150
gis.generalisation.clip.area.multiple=3

# Spatial Indexing
index.spatial.storage.pagesize=512
index.spatial.rtree.maxnodecapacity=12
index.spatial.rtree.minnodecapacity=5
index.spatial.rtree.nodesplit=Quadratic
index.spatial.cache.size=8

# Visibility Algorithms
gis.projection.depth=MinDistanceDepthComparator
gis.projection.visibility.viewfrustum=ViewFrustumCullingFilter
gis.projection.visibility.occlusion=BackfaceCulling
```

A.2 Benchmark Harness

Listing A.2: The `Harness` implementation

```
Benchmark[] benchmarks = new Benchmark[12];
benchmarks[0] = new StartupBenchmark();
benchmarks[1] = new IOSpeedBenchmark();
benchmarks[2] = new MultiscaleBenchmark();
benchmarks[3] = new IndexingBenchmark();
benchmarks[4] = new ClippingBenchmark();
benchmarks[5] = new SimplificationBenchmark();
benchmarks[6] = new LODBenchmark();
benchmarks[7] = new ProjectionBenchmark();
benchmarks[8] = new DepthComparatorBenchmark();
benchmarks[9] = new VisibilityBenchmark();
benchmarks[10] = new ViewFrustumCullingBenchmark();
benchmarks[11] = new UIResponsivenessBenchmark();

Reporter.init();

for (int i = 0; i < benchmarks.length; i++) {

        StatisticSummary.reset();
        // Get number of repetitions from config file
        int repeat = MobileGIS..getProperty(Harness.
            REPEAT_BENCHAMRKS_PROPERTY);

        for (int j = 0; j < repeat; j++) {
                Reporter.reset();
                benchmarks[i].start();
                while (benchmarks[i].isAlive()) Thread.sleep(500)
                    ;
                StatisticSummary.addSample(Reporter.getReport());
        }
        MobileGIS.log.debug(StatisticSummary.getSummary(), this);
}
```

A.3 Energy Evaluation User Simulator

Listing A.3: The sequence of simulated user events

```
// Repeat set of events five times
for (int i = 0; i < 5; i++) {
        map.handleEvent(new PanEvent(this, PanEvent.EAST, 3));
        waitForClientIdle();
        map.handleEvent(new PanEvent(this, PanEvent.EAST, 3));
        waitForClientIdle();
        map.handleEvent(new PanEvent(this, PanEvent.WEST, 3));
        waitForClientIdle();
        map.handleEvent(new PanEvent(this, PanEvent.WEST, 3));
        waitForClientIdle();
        map.handleEvent(new PanEvent(this, PanEvent.NORTH, 3));
        waitForClientIdle();
        map.handleEvent(new PanEvent(this, PanEvent.SOUTH, 3));
        waitForClientIdle();
        map.handleEvent(new PanEvent(this, PanEvent.SOUTH, 3));
        waitForClientIdle();
        map.handleEvent(new PanEvent(this, PanEvent.NORTH, 3));
        waitForClientIdle();
        map.handleEvent(new ZoomEvent(this, ZoomEvent.ABSOLUTE,
            map.getScale().getRangeAbove());
        waitForClientIdle();
        map.handleEvent(new ZoomEvent(this, ZoomEvent.ABSOLUTE,
            map.getScale().getRangeAbove());
        waitForClientIdle();
        map.handleEvent(new ZoomEvent(this, ZoomEvent.ABSOLUTE,
            map.getScale().getRangeBelow());
        waitForClientIdle();
        map.handleEvent(new ZoomEvent(this, ZoomEvent.ABSOLUTE,
            map.getScale().getRangeBelow());
        waitForClientIdle();
}
```

A.4 Screenshots

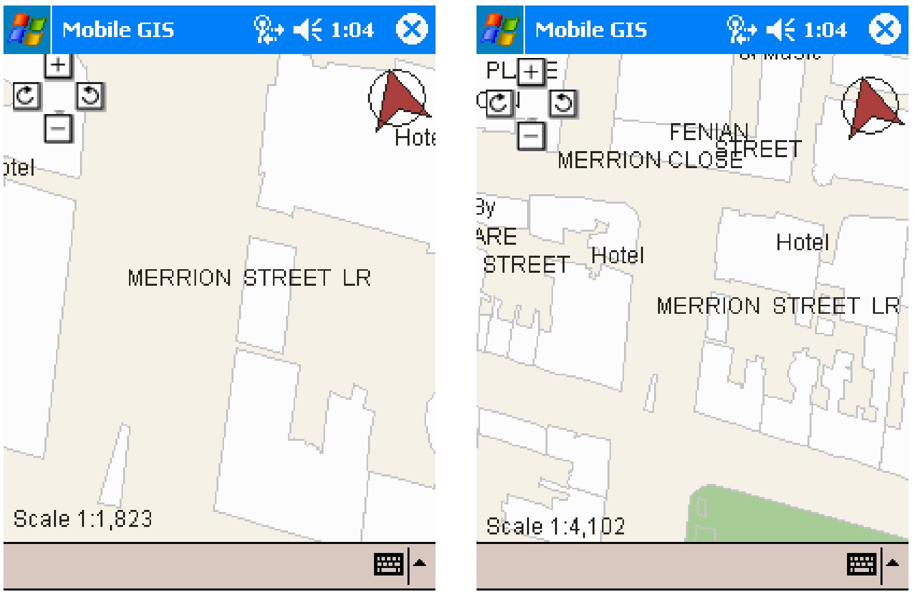


Figure A.1: Zoom map interface screenshot

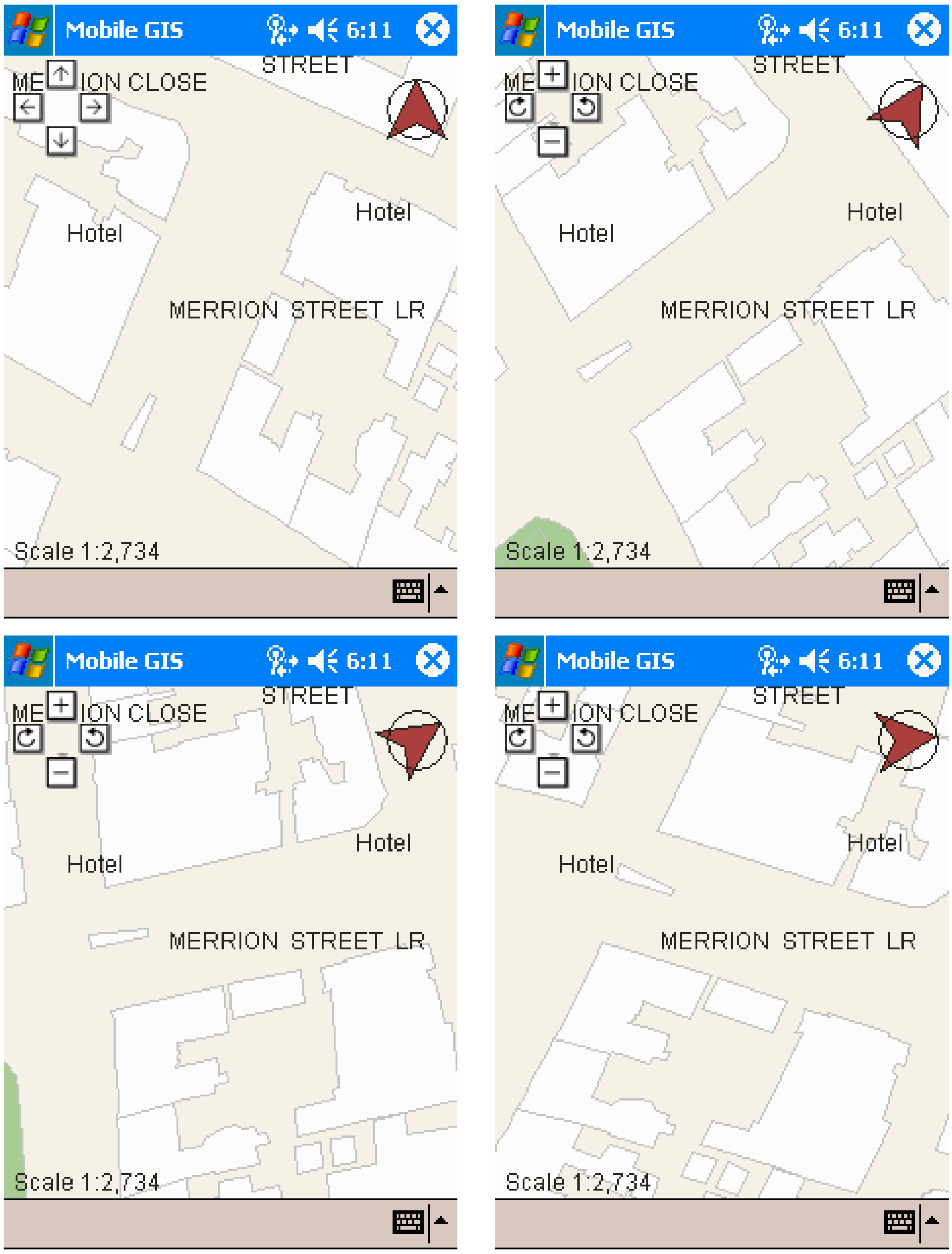


Figure A.2: Rotating map interface screenshot (rotating clockwise from top left)

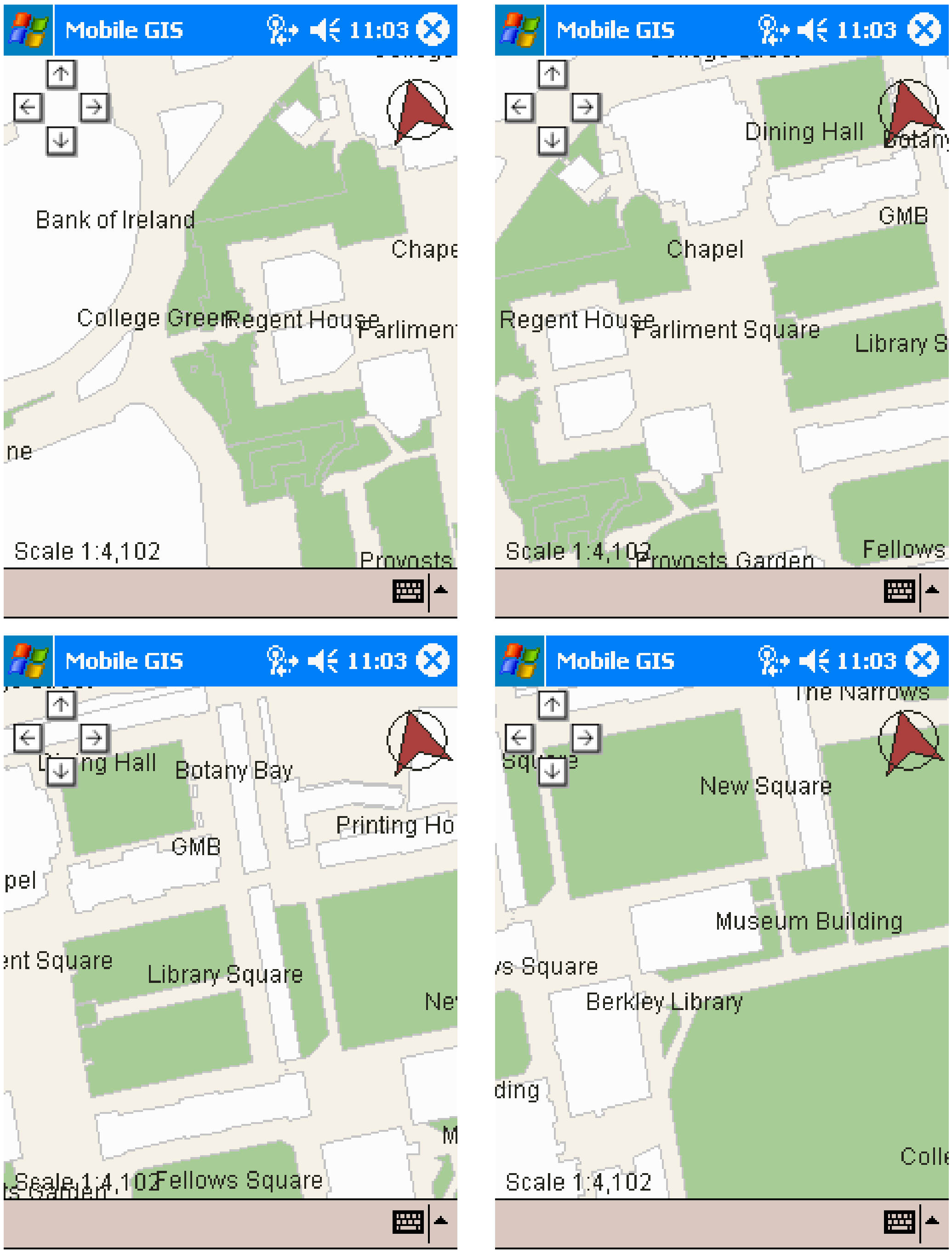
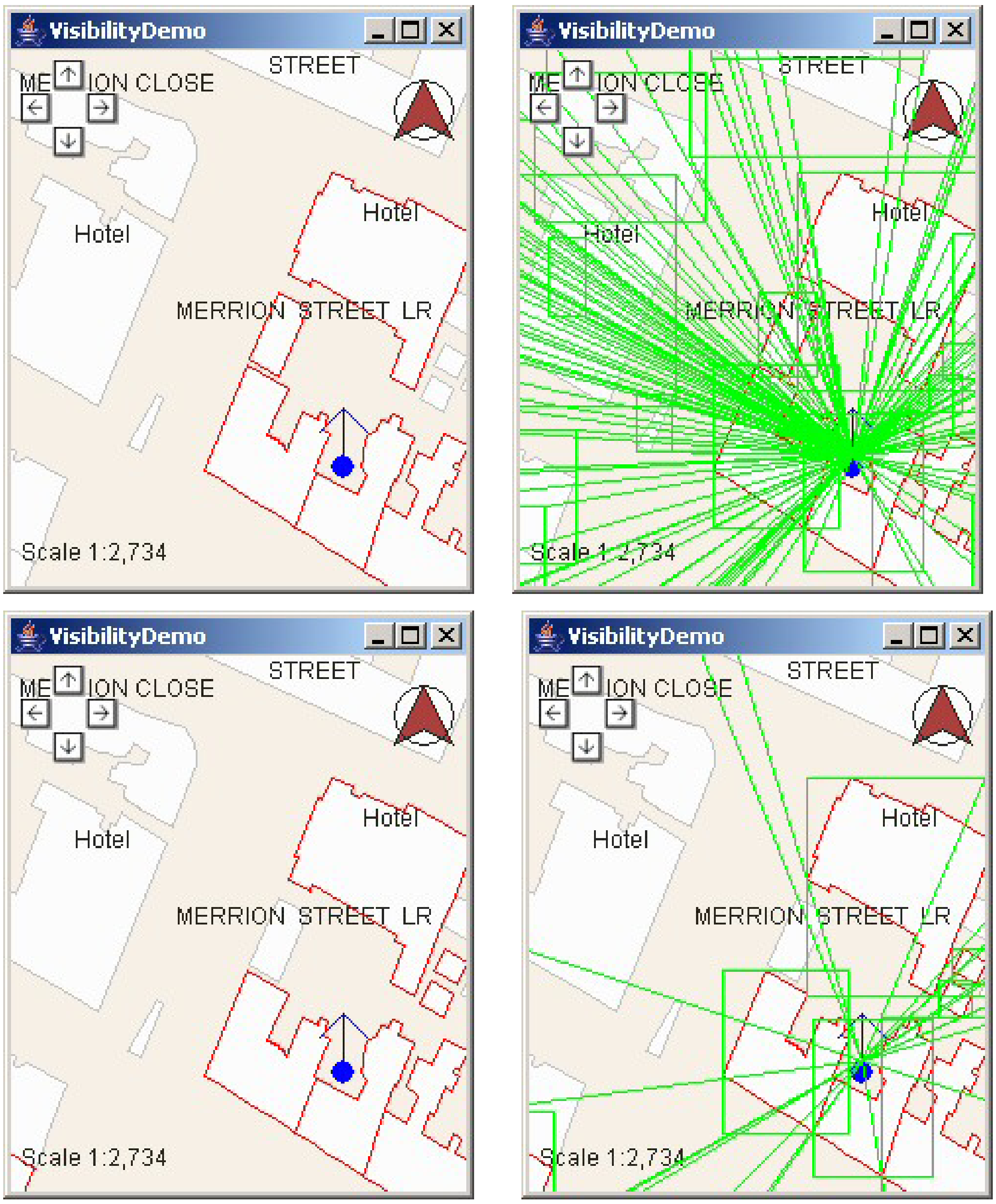


Figure A.3: Pan map interface screenshot



(a) fgfg

Figure A.4: Visibility determination accuracy

Appendix B

Source Code Measurements

Package	Classes	Functions	NCSS	Javadocs
hermes.context	1	1	9	2
hermes.context.location	8	53	281	52
hermes.context.location.gps	1	2	30	3
hermes.gis	4	40	525	33
hermes.gis.asynch	3	8	58	13
hermes.gis.event	14	45	282	48
hermes.gis.generalisation.clipping	10	35	363	21
hermes.gis.generalisation.elimination	3	14	69	9
hermes.gis.generalisation.simplification	5	35	199	18
hermes.gis.geometry	14	126	834	100
hermes.gis.graph	8	52	298	36
hermes.gis.graph.traverse	7	24	192	25
hermes.gis.index	3	15	57	16
hermes.gis.index.spatial	7	81	624	68
hermes.gis.io	18	290	2270	232
hermes.gis.io.filefilter	1	3	73	4
hermes.gis.io.multiscale	3	41	263	22
hermes.gis.layer	12	57	561	51
hermes.gis.layer.openstreet	1	2	10	2
hermes.gis.math	3	62	306	64

Package	Classes	Functions	NCSS	Javadocs
hermes.gis.net	5	17	562	20
hermes.gis.projection	6	77	321	44
hermes.gis.projection.model	1	10	54	11
hermes.gis.query	11	62	487	59
hermes.gis.render	1	1	44	1
hermes.gis.tools	7	31	247	32
hermes.gis.tools.benchmark	22	66	1068	57
hermes.gis.tools.test	6	15	153	13
hermes.gis.ui	4	19	190	14
hermes.gis.ui.event	1	2	24	2
hermes.gis.ui.style	2	26	76	16
hermes.gis.visibility	11	42	353	35
Total	203	1354	10882	1123

Table B.1: Implementation NCSS per package

Appendix C

Glossary

To avoid misunderstandings a list of often used terms, abbreviations and symbols in this thesis are defined.

C.1 Terminology

FoV Field of view (also field of vision), the angular extent of the observable world.

Graticule A network of crossing lines on a map, representing parallels and meridians as defined by the projection.

I/O Input / Output. Used to refer to reading and writing to memory, storage or network.

Generalisation A process whereby spatial objects can be depicted on smaller scales through simplification, omission and combination.

LRU Least Recently Used.

Model-oriented Generalisation The process of reducing geometry detail as a consequence of reducing scale at the spatial data model level.

MANET Mobile ad-hoc network, a mobile network, where stations can move around and change the network topology.

MBR Minimum bounding rectangle (or volume), also known as bounding box or envelope, is an expression of the maximum extents of a 2-dimensional object.

MRDB Multiple Representation Database.

NIC Network Interface Card.

NCSS Non-commented Source Statements.

Occlude To visually obstruct.

OGC The Open GeoSpatial Consortium: an organization that is developing standards for geospatial and location based services.

OSI Ordnance Survey Ireland.

P2P Peer-to-peer: A communications model in which each party has the same capabilities.

PoV Point of view, a combination of position, orientation and field of view.

Raster A method of storing image data which consists of cells (pixels) which make up rows and columns.

Shapefile ESRI vector file format for non-topological spatial and attribute data.

Spatially-aware An application is spatially-aware if it contains a model of the user's environment and uses this model to infer relationships between real world objects. Spatially-aware applications will generally incorporate other sources of context gathered from sensors into their environment models.

Vector Vector graphics consist of a collection of geometric shapes.

View Frustum The area between two vertical planes where each plane corresponds to the limit of field of view.

C.2 Symbols

E A spatial extent.

$E^{n,m}$ containing geometry for n shapes consisting of m coordinates, the maximal representation $E^{n,m}$.

LoD Level of detail.

LoD_0 The most detailed level of detail.

mA milliampere: one thousandth of an ampere (a measure of flow of electric current).

mC millicoulomb: The coulomb (symbol: C) is the *International System of Units* unit of electric charge.

mAh milliampere hour: A measure of a battery's total capacity.

px Pixel

s Scale.

SR_i Scale range where i is the midpoint of the range

t_1 Timer 1

Appendix D

API

TODO: Introduce section and explain what has been included / omitted. Also explain how this documentation was generated.

Visibility skipped + testing, benchmarking, case study applications

listed by package (ref package structure diagram Figure 4.2)

only public, not showing inheritance

D.1 Package ie.tcd.cs.dsg.hermes.gis.generalisation.simplification

Interfaces

Interface **Simplify2D**

Classes implementing this interface are line simplification (generalisation) algorithms.
http://www.sli.unimelb.edu.au/gisweb/LGmodule/LGSimplification.htm
Views on a geometry table with simplified geometries work well with scale dependent layering for viewing data. The function is very quick, and means the viewing software can have several orders of magnitude less vertices to render...

Fields

- public static final int DEFAULT_SIMPLIFICATION_TOLERANCE
  - The minimum number of pixels lines should be after simplification
- public static final String SIMPLIFICATION_THRESHOLD_PROPERTY
  - The key for the property defining the threshold that complex simplification can be used at
- public static final String SIMPLIFICATION_TOLERANCE_PROPERTY
  - The key for the property that specifies the tolerance in pixels
- public static final int SIMPLE
  - Refers to the simple vertex reduction algorithm for coastlines
- public static final int COMPLEX
  - Refers to the complex vertex reduction algorithm for buildings

Methods

`public Polygon` **simplify**`( Polygon` **poly**`, float` **scale** `)`

- **Usage**

  - Simplifies the specified polygon using the current algorithm implementation and the parameters passed to its constructor.

- **Parameters**
  - `poly` - the polygon to simplify (No guarantees about topological correctness are made)
  - `scale` - the scale parameter determines how much the shape is simplified
- **Returns** - a modification of the specified line or new object

`public Polyline` **simplify**`( Polyline` **line**`, float` **scale** `)`

- **Usage**
  - Simplifies the specified polyline using the current algorithm implementation and the parameters passed to its constructor.
- **Parameters**
  - `line` - the line to simplify (No guarantees about topological correctness are made)
  - `scale` - the scale parameter determines how much the shape is simplified
- **Returns** - a modification of the specified line or new object

Classes

Class **DouglasPeucker**

An implementation of the Douglas Peuker Polyline Simplification (Vertex Reduction) Algorithm. The Douglas-Peucker algorithm has a sophisticated way of skipping points. It basically visualizes long lines, and discards points that are close to the virtual lines. This is more calculation intensive than the nth point algorithm, but provides much better results. David Douglas & Thomas Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature", The Canadian Cartographer 10(2), 112-122 (1973)
John Hershberger & Jack Snoeyink, "Speeding Up the Douglas-Peucker Line-Simplification Algorithm", Proc 5th Symp on Data Handling, 134-143 (1992). UBC Tech Report available online from NEC ResearchIndex.

extends java.lang.Object

implements Simplify2D, ie.tcd.cs.dsg.hermes.gis.tools.Algorithm

FIELDS

- public static int DEFAULT_MIN_POINTS
  - Default minimum number of points required to represent polygon

CONSTRUCTORS

`public` **DouglasPeucker**`( int` **pixTolerance** `)`

- **Parameters**
  - `pixTolerance` - the tolerance in pixels

METHODS

`public String` **getName**`( )`

`public Polygon` **simplify**`( Polygon` **poly**`, float` **scale** `)`

`public Polyline` **simplify**`( Polyline` **line**`, float` **scale** `)`

CLASS **KraakOrmeling**

Implementation of nth Point algorithm as proposed by Kraak and Ormeling. The nth point algorithm consists of skipping every n points. This is quick and easy, but may miss corners and other important details.

extends java.lang.Object

implements Simplify2D, ie.tcd.cs.dsg.hermes.gis.tools.Algorithm

FIELDS

- public static int DEFAULT_MIN_POINTS

CONSTRUCTORS

`public` **KraakOrmeling**`( int` **nthPoint** `)`

- **Parameters**
  - `nthPoint` - This parameter is not pixel tolerance as with other simplification algorithms

METHODS

`public String` **getName**`( )`

`public void` **setMinPoints**`( int` **i** `)`

`public Polygon` **simplify**`( Polygon` **poly**`, float` **scale** `)`

`public Polyline` **simplify**`( Polyline` **line**`, float` **scale** `)`

CLASS **ShapeSimplify**

Simplifys geometry in a configurable CPU aware manner.

extends java.lang.Object

implements Simplify2D, ie.tcd.cs.dsg.hermes.gis.tools.Algorithm

CONSTRUCTORS

`public` **ShapeSimplify**`( )`

- **Usage**
  - Constructor. Loads parameters from properties file

`public` **ShapeSimplify**`( int` **threshold**`, int` **tolerance** `)`

- **Usage**
  - Constructor. Creates a simplification object with the supplied parameters.
- **Parameters**

  – `threshold` - the number of points at which we switch from simple to complex generalisation
  – `tolerance` - the pixTolerance or Nth point algorithm parameter

Methods

`public String` **getName**`( )`

`public Polygon` **simplify**`( Polygon` **poly**`, float` **scale** `)`

`public Polyline` **simplify**`( Polyline` **line**`, float` **scale** `)`

D.2 Package ie.tcd.cs.dsg.hermes.gis.io

Interfaces

INTERFACE **GISDataSource**

Basic DataSource operations.

implements ie.tcd.cs.dsg.hermes.gis.tools.benchmark.IOAccounting

FIELDS

- public static final String COUNT_PROPERTY

  – Property key for number of data sources

Methods

`public boolean` **canRead**`( )`

- **Usage**
  - Tests to see if you have permission to read from this DataSource. Could be based on underlying file permissions of just set in the constructor of a DataSource implementation.
- **Returns** - true if the read permission is stored in _premissions list

`public boolean` **canWrite**`( )`

- **Usage**
  - Tests to see if you have permission to write to this DataSource. Could be based on underlying file permissions of just set in the constructor of a DataSource implementation.
- **Returns** - true if the write permission is stored in _premissions list

`public String` **getName**`( )`

- **Usage**
  - Returns a readable name to identify a data source. Should be unique across the different data sources in use. I would suggest using the file name for any file based data sources and the host / database name for databases.
- **Returns** - a String that can be used to identify this data source

Interface **SpatialDataSource**

SpatialDataSource is the interface to a readable and writable source of geometry and its associated attributed. Implementers of this class may provide access to geometry from database's or ESRI shapefiles.

implements GISDataSource

Fields

- public static final String MULTISCALE_DATASOURCE_PROPERTY
  - The property specifying whether to use a multiscale datasource
- public static final String SPATIAL_INDEX_PROPERTY
  - The property specifying the type of spatial index to use

Methods

`public void` **addGeometry**`( ShapeList` **shapeRecords**`, Rectangle` **extent** `)`

- **Usage**
  - Adds new geometry to the data source and updates index's. It is assumed that the Shapes share the same coordinate system as the existing Shapes stored in the datasource and are the same type of Shape.

    Note. Single shapes or multipart shapes only. i.e. adding multiple points at once will cause an exception.
- **Parameters**
  - `shapeRecords` - the Shapes to add and their record numbers
  - `extent` - The MBR of the data contained in the records

`public ShapeList` **getAttributesForRecords**`( ShapeList` **records** `)`

- **Usage**
  - Adds the attributes for the Shapes to the ShapeRecords
- **Parameters**
  - `records` - the records to get attributes for
- **Returns** - the same records with attributes added

`public ShapeList` **getGeometryForRecords**`( ShapeList` **records** `)`

- **Usage**
  - Reads geometry for specified records
- **Parameters**
  - `records` - The index data retrieved by getRecords()
- **Returns** - ShapeList A list of geometry

`public SpatialIndex` **getIndex**`( )`

- **Usage**
  - Returns the spatial index in use by this data source. Each source maintains it's own index and this methos is used by benchamrking code to get the indexing algorithm in use by each data source for purposes of comparison.
- **Returns** - the spatial index in use

`public ShapeList` **getRecords**`( Rectangle` **area** `)`

- **Usage**
  - Reads geometry record index's to retrieve records that intersect the minimum bounding rectangle specified. These records are later used to retrieve the actual geometry.
- **Parameters**
  - `area` - The minimum bounding rectangle
- **Returns** - ShapeList A list of geometry record numbers

`public ScaleRange` **getScale**`( )`

- **Usage**
  - Each data source is assumed to supply shape data for a fixed resolution. This method returns the minimum scale at which data from this source should be displayed.
- **Returns** - the recommended minimum scale at which this data should be rendered. Below this scale the level of detail becomes too low.

`public int` **getShapeType**`( )`

- **Usage**
  - Returns the type of shape this data source supplies. Assumes that each data source can contain only one type of shape which is the case with ESRI Shapefiles.
- **Returns** - the shape type (See `ShapeConstants`)

`public boolean` **hasGeometry**`( )`

- **Usage**
  - Tests to see if the data source has geometry or is empty. Used to check for new or empty data stores in order to avoid querying files that don't contain data.
- **Returns** - true is the data source contains shape geometry

`public void` **removeGeometry**`( ShapeRecord` **record** `)`

- **Usage**
  - Deletes the geometry for the data source and removes all references to the specified shape from the index's.
- **Parameters**
  - `record` - the record to remove

`public void` **setScale**`( ScaleRange` **scale** `)`

- **Usage**
  - Each data source is assumed to supply shape data for a fixed resolution. This method sets the scale at which data from this source can be displayed.
- **Parameters**
  - `scale` - a scale range specifying the min & max scales

Classes

CLASS **AbstractShapeDataSource**

Abstracts the geometry caching and data source permissions so they don't need to be implemented in every DataSource implementation.

extends java.lang.Object

implements SpatialDataSource

CONSTRUCTORS

`public` **AbstractShapeDataSource**`( )`

METHODS

`public void` **addGeometry**`( ShapeList` **shapeRecords**`, Rectangle` **extent** `)`

`public boolean` **canRead**`( )`

`public boolean` **canWrite**`( )`

`public void` **clearRecordCache**`( )`

`public abstract ShapeList` **getAttributesForRecords**`( ShapeList` **records** `)`

`public ShapeList` **getRecordCache**`( )`

`public void` **removeGeometry**`( ShapeRecord` **record** `)`

`public void` **setRecordCache**`( ShapeList` **list** `)`

`public void` **setScale**`( ScaleRange` **scale** `)`

CLASS **DBFFile**

Enables the user to stored data store herein to be saved to a file conforming to the DBF III file format specification.

DBF Specification Available at:

http://www.cs.cornell.edu/Courses/cs212/2001fa/Project/Part1/dbf.htm

extends ie.tcd.cs.dsg.hermes.gis.io.LittleIndianFile

Fields

- public static final int TYPE_CHARACTER
- public static final int TYPE_DATE
- public static final int TYPE_NUMERIC
- public static final int TYPE_LOGICAL
- public static final int TYPE_MEMO

Constructors

`public` **DBFFile**`( File` **f** `)`

- **Usage**
  - Reads the database file into memory or opens an empty file for writing DBD data to.
- **Parameters**
  - `f` - a DBase file or empty file

`public` **DBFFile**`( File` **f**`, int` **columnCount** `)`

- **Usage**
  - Creates a blank DbfFile
- **Parameters**
  - `columnCount` - The number of columns this model will manage
  - `f` - a DBase file or empty file

Methods

`public void` **addRow**`( Object [] ` **columns** `)`

- **Usage**
  - Adds a row of data to the the model
- **Parameters**
  - `columns` - A collection of columns that comprise the row of data

`public void` **deleteRow**`( int ` **rowNumber** `)`

- **Usage**
  - Deletes the specified row from the database table. Actuallt the record number (indexed from 1) is expected. The row at position record - 1 will be deleted from memory and disk.
- **Parameters**
  - `rowNumber` -

`public int` **getColumnCount**`( )`

- **Usage**
  - Retrieves the number of columns that exist in the model
- **Returns** - The number of columns that exist in the model

`public int` **getColumnIndex**`( String ` **name** `)`

- **Usage**
  - Returns the column number matching the specified name
- **Parameters**
  - `name` - the column name
- **Returns** - the column number or -1 if the column does not exist

`public String` **getColumnName**`( int ` **column** `)`

- **Usage**

  – Retrieves the column name for the passed in column index

- **Parameters**
  – `column` - The column index
- **Returns** - The column name for the given column index

`public String` **getColumnNames**`( )`

- **Returns** - an array of the columnnames

`public Object` **getRow**`( int` **rowNumber** `)`

- **Usage**
  – Gets an entire row of the table
- **Parameters**
  – `rowNumber` -
- **Returns** - the contents of the row

`public int` **getRowCount**`( )`

- **Usage**
  – Retrieves the number of rows that exist in the model
- **Returns** - The number rows that exist in the model

`public Object` **getValueAt**`( int` **row**`, int` **column** `)`

- **Usage**
  – Retrieves a value for a specific column and row index
- **Returns** - Object A value for a specific column and row index

`public void` **readData**`( )`

- **Usage**
  – Reads the data and places data in a class scope ArrayList of records

`public void` **readHeader**`( )`

`public void` **setColumnName**`( int` **column**`, String` **name** `)`

- **Usage**
  - Sets the column name for the passed-in field index
- **Parameters**
  - `column` - The column index
  - `name` - The name to assign for the passed-in column index

`public void` **setType**`( int` **column**`, byte` **type** `)`

- **Usage**
  - Sets the column type for the passed-in field index
- **Parameters**
  - `column` - The column index
  - `type` - The type of column to assign for the passed-in column index

`public void` **setValueAt**`( Object` **object**`, int` **row**`, int` **column** `)`

- **Usage**
  - Sets the value at the specified row and column in the table
- **Parameters**
  - `object` - the value
  - `row` - indexed from 0
  - `column` -

Class **ESRIFile**

Includes some of the functionality that is common to all files that make up a Shapefile. e.g. SSX, SHP and SHX files all share the same format file header. This class reads and parses that header.

extends ie.tcd.cs.dsg.hermes.gis.io.LittleIndianFile

Fields

- public static final float DEFAULT_MIN_SCALE
  - The default value for files not specifying a minimum scale

Constructors

`public` **ESRIFile**`( File` **f** `)`

- **Parameters**
  - `f` -

Methods

`public Rectangle` **getBounds**`( )`

- **Returns** - the bounds of the geometry in the file

`public int` **getFileLength**`( )`

- **Returns** - the length of the file

`public final ScaleRange` **getScale**`( )`

- **Returns** - the scale of the data in the file (from header bytes)

`public int` **getShapeType**`( )`

- **Returns** - the type of geometry in the file

`public byte` **readHeader**`( )`

- **Usage**
  - Reads the file header and returns it as a byte array
- **Returns** - the file header

`public IndexEntry` **readIndexRecord**`( byte [] `**b**`, int `**off**`, int `**recordNumber**`, int `**shapeType** `)`

`public IndexEntry` **readSpatialIndexRecord**`( int `**recordNumber**`, int `**shapeType** `)`

`public void` **setScale**`( ScaleRange `**range** `)`

`public void` **setShapeType**`( int `**i** `)`

`public void` **writeHeader**`( )`

- **Usage**
  - Writes the specified header bytes to the beginning of the file

CLASS **ESRIShapeFile**

An ESRI shapefile consists of a main file, an index file, and a dBASE table. The main file is a direct access, variable-record-length file in which each record describes a shape with a list of its vertices. In the index file, each record contains the offset of the corresponding main file record from the beginning of the main file. The dBASE table contains feature attributes with one record per feature. The one-to-one relationship between geometry and attributes is based on record number. Attribute records in the dBASE file must be in the same order as records in the main file.

extends ie.tcd.cs.dsg.hermes.gis.io.AbstractShapeDataSource

CONSTRUCTORS

`public` **ESRIShapeFile**`( File `**shpFile**`, DBFFile `**database**`, int `**shapeType** `)`

- **Usage**
  - Creates a shapefile with the specified file components. This constructor is used to create shapefiles that share the same database file.
- **Parameters**
  - `database` - ESRI Shape file DB2 database file

`public` **ESRIShapeFile**`( SHPFile` **shpFile**`, DBFFile` **database**`, SpatialIndex` **index** `)`

- **Usage**
  - Creates a shapefile with the specified file components. This constructor is used to create the master shapefile in MultiScale Shapefiles in order to allow the creation of possibly missing SSX files.
- **Parameters**
  - `shpFile` - ESRI Shape file main geometry file
  - `database` - ESRI Shape file DB2 database file
  - `index` - ESRI Spatial Index file

`public` **ESRIShapeFile**`( String` **shpFileName** `)`

- **Usage**
  - Opens all the files associated with a Shapefile. If the main file does not exist then new files are created. If index's are missing they are created from the main shape file.
- **Parameters**
  - `shpFileName` - the path and filename of the main shape file

Methods

`public final void` **addGeometry**`( ShapeList` **shapeRecords**`, Rectangle` **extent** `)`

- **Usage**
  - See the general contract of the `addGeometry` method of `SpatialDataSource`.

`public final ShapeList` **getAttributesForRecords**`( ShapeList` **records** `)`

`public DBFFile` **getDatabase**`( )`

- **Returns** - the database file

`public final Rectangle` **getExtents**`( )`

- **Usage**
  - See the general contract of the `getExtents` method of `SpatialDataSource`.
- **See Also**
  - `ie.tcd.cs.dsg.hermes.gis.io.SpatialDataSource.getExtents()`

`public final ShapeList` **getGeometryForRecords**`( ShapeList` **records** `)`

`public SpatialIndex` **getIndex**`( )`

- **Returns** - the index file

`public final String` **getName**`( )`

`public File` **getParent**`( )`

- **Usage**
  - This method returns a `String` the represents this file's parent. `null` is returned if the file has no parent. The parent is determined via a simple operation which removes the
- **Returns** - The parent directory of this file

`public final ShapeList` **getRecords**`( Rectangle` **area** `)`

- **Usage**
  - See the general contract of the `getRecords` method of `SpatialDataSource`.
- **See Also**
  - `ie.tcd.cs.dsg.hermes.gis.io.SpatialDataSource.getRecords(Rectangle)`

`public ScaleRange` **getScale**`( )`

`public final int` **getShapeType**`( )`

- **Usage**

  - See the general contract of the `getShapeType` method of `SpatialDataSource`.
- **See Also**
  - `ie.tcd.cs.dsg.hermes.gis.io.SpatialDataSource.getShapeType()`

`public final boolean` **hasGeometry**`( )`

`public final void` **removeGeometry**`( ShapeRecord` **record** `)`

- **Usage**
  - See the general contract of the `removeGeometry` method of `SpatialDataSource`.
- **See Also**
  -

    `ie.tcd.cs.dsg.hermes.gis.io.SpatialDataSource.removeGeometry(ShapeRecord`

`public final void` **setScale**`( ScaleRange` **scale** `)`

Class **LittleIndianFile**

Provides Number format independent read/write access to a the specified binary file.

extends java.lang.Object

implements ie.tcd.cs.dsg.hermes.gis.geometry.ShapeConstants

Constructors

`public` **LittleIndianFile**`( File` **f** `)`

- **Parameters**
  - `f` - the file to read / write from.

Methods

`public final long` **length**`( )`

- **Usage**
  - Returns the length of this file.
- **Returns** - the length of this file, measured in bytes.

`public final int` **readBEInt**`( )`

- **Usage**
  - Reads a big endian integer from a little endian input stream.
- **Returns** - the int read from the inputstream

`public final int` **readBEInt**`( byte [] b, int` **off** `)`

- **Usage**
  - Reads a big endian integer.
- **Parameters**
  - `b` - the raw data buffer
  - `off` - the offset into the buffer where the int resides
- **Returns** - the int read from the buffer at the offset location

`public final Rectangle` **readBox**`( )`

- **Usage**
  - Reads a bounding box record. A bounding box is four double representing, in order, xmin, ymin, xmax, ymax.
- **Returns** - the box read from the buffer at the offset location

`public final Rectangle` **readBox**`( byte [] b, int` **off** `)`

- **Usage**
  - Reads a bounding box record. A bounding box is four double representing, in order, xmin, ymin, xmax, ymax.
- **Parameters**
  - `b` - the raw data buffer
  - `off` - the offset into the buffer where the int resides

- **Returns** - the point read from the buffer at the offset location

`public final byte` **readByte**`( )`

- **Usage**
  - Reads and returns one input byte. The byte is treated as a signed value in the range -128 through 127, inclusive.
- **Returns** - the 8-bit value read.

`public final IndexEntry` **readIndexEntryData**`( )`

- **Usage**
  - Read an Index Entry from disk.

    Used by R-Tree Spatial Index's which store the bounding box as part of the surrounding tree data structure.
- **Returns** - the IndexEntry that was read

`public IndexEntry` **readIndexRecord**`( byte [] ` **b**`, int ` **off**`, int` **recordNumber**`, int ` **shapeType** `)`

- **Usage**
  - Reads an Index record from the specified byte array;
- **Parameters**
  - `b` - the raw data buffer
  - `off` - the offset into the buffer where the double resides
- **Returns** - the content length and offset as an IndexRecord

`public final IndexEntry` **readIndexRecord**`( int ` **recordNumber**`, int` **shapeType** `)`

- **Usage**
  - Reads an IndexRecord record.
- **Returns** - the IndexRecord read from the file

`public final double` **readLEDouble**`( )`

- **Usage**
  - Reads a little endian double into a big endian double
- **Returns** - double A big endian double

`public final double` **readLEDouble**`( byte [] b, int` **off** `)`

- **Usage**
  - Reads a little endian double.
- **Parameters**
  - `b` - the raw data buffer
  - `off` - the offset into the buffer where the double resides
- **Returns** - the double read from the buffer at the offset location

`public final float` **readLEFloats**`( float [] f, int` **numberToRead** `)`

- **Usage**
  - Reads the specified number of little endian doubles into a big endian double, casts them to floats and assigns the floats to the specified array.
- **Parameters**
  - `f` - an array to fill with the read float values
  - `numberToRead` - the number of float values to read
- **Returns** - the array specified

`public final int` **readLEInt**`( )`

- **Usage**
  - Translates a little endian int into a big endian int
- **Returns** - int A big endian int

`public final int` **readLEInt**`( byte [] b, int` **off** `)`

- **Usage**
  - Translates a little endian int into a big endian int

- **Returns** - int A big endian int

`public final long` **readLELong**`( )`

- **Usage**
  - Reads a little endian double into a big endian double
- **Returns** - double A big endian double

`public final long` **readLELong**`( byte [] b, int` **off** `)`

- **Usage**
  - Reads a little endian 8 byte integer.
- **Parameters**
  - `b` - the raw data buffer
  - `off` - the offset into the buffer where the long resides
- **Returns** - the long read from the buffer at the offset location

`public final short` **readLEShort**`( )`

- **Usage**
  - Translates little endian short to big endian short
- **Returns** - short A big endian short

`public final short` **readLEShort**`( byte [] b, int` **off** `)`

- **Usage**
  - Translates little endian short to big endian short
- **Parameters**
  - `b` - the raw data buffer
  - `off` - the offset into the buffer where the int resides
- **Returns** - short A big endian short

`public final Point` **readPoint**`( )`

- **Usage**

  - Reads a point record. A point record is a double representing the x value and a double representing a y value.
- **Returns** - the point read from the buffer at the offset location

`public final Point` **readPoint**`( byte [] `**b**`, int `**off**` )`

- **Usage**
  - Reads a point record. A point record is a double representing the x value and a double representing a y value.
- **Parameters**
  - `b` - the raw data buffer
  - `off` - the offset into the buffer where the int resides
- **Returns** - the point read from the buffer at the offset location

`public IndexEntry` **readSpatialIndexRecord**`( int `**recordNumber**`, int `**shapeType**` )`

- **Usage**
  - Reads a SpatialIndexRecord from the file
- **Parameters**
  - `recordNumber` - the record number of the index record being read. Used to keep track of where the index record was locaten in the file
- **Returns** - the SpatialIndexRecord read

`public final String` **readString**`( int `**length**` )`

- **Usage**
  - Constructs a string from the underlying input stream
- **Parameters**
  - `length` - The length of bytes to read

`public final int` **readUnsignedByte**`( )`

- **Usage**

  - Reads one input byte, zero-extends it to type int, and returns the result, which is therefore in the range 0 through 255.

- **Returns** - the unsigned 8-bit value read.

`public final void` **writeBEInt**`( byte [] `**buffer**`, int `**off**`, int `**v**` )`

- **Usage**
  - Writes a number of type int in little endian
- **Parameters**
  - `buffer` - the raw data buffer
  - `off` - the offset into the buffer where the int resides
  - `v` - A number of type int

`public final void` **writeBEInt**`( int `**v**` )`

- **Usage**
  - Writes a number of type int in little endian
- **Parameters**
  - `v` - A number of type int

`public final void` **writeBox**`( byte [] `**b**`, int `**off**`, Rectangle `**box**` )`

- **Usage**
  - Writes the given bounding box to the given buffer at the given location. The bounding box is written as four doubles representing, in order, xmin, ymin, xmax, ymax.
- **Parameters**
  - `b` - the data buffer
  - `off` - the offset into the buffer where writing should occur
  - `box` - the bounding box to write

`public final void` **writeBox**`( Rectangle `**box**` )`

- **Usage**

  - Writes the given bounding box to the file as four doubles representing, in order, xmin, ymin, xmax, ymax.

- **Parameters**
  - `box` - the bounding box to write

`public final void` **writeByte**`( int` **v** `)`

- **Usage**
  - Writes out a byte to the underlying output stream as a 1-byte value. If no exception is thrown, the counter written is incremented by 1.
- **Parameters**
  - `v` - a byte value to be written.

`public final void` **writeIndexEntryData**`( IndexEntry` **entry** `)`

- **Usage**
  - Writes the specified IndexEntry to the underlying file at the current file pointer

    Used to write spatial index data to R-Tee based index.
- **Parameters**
  - `entry` -

`public final void` **writeIndexRecord**`( byte [] b, int` **off**`, IndexEntry` **record** `)`

- **Usage**
  - Writes an Index record from the specified byte array;
- **Parameters**
  - `b` - the raw data buffer
  - `off` - the offset into the buffer where the double resides
  - `record` - an IndexRecord

`public final void` **writeIndexRecord**`( IndexEntry` **record** `)`

- **Usage**
  - Writes an IndexRecord record. Ensure index record is written to the correct byte offset in the index file in order to reflect the original record number
- **Parameters**
  - `record` - an IndexRecord

`public final void` **writeLEDouble**`( byte [] `**buffer**`, int `**off**`, double `**d**` )`

- **Usage**
  - Writes a number a number of type double in little endian
- **Parameters**
  - `buffer` - the raw data buffer
  - `off` - the offset into the buffer where the double resides
  - `d` - A number of type double

`public final void` **writeLEDouble**`( double `**d**` )`

- **Usage**
  - Writes a number a number of type double in little endian
- **Parameters**
  - `d` - A number of type double

`public final void` **writeLEInt**`( byte [] `**buffer**`, int `**off**`, int `**i**` )`

- **Usage**
  - Writes a number of type int in little endian
- **Parameters**
  - `buffer` - the raw data buffer
  - `off` - the offset into the buffer where the int resides
  - `i` - A number of type int

`public final void` **writeLEInt**`( int `**i**` )`

- **Usage**
  - Writes a number of type int in little endian
- **Parameters**
  - `i` - A number of type int

`public final void` **writeLELong**`( byte [] `**buffer**`, int `**off**`, long `**l**` )`

- **Usage**
  - Writes a number of type long in little endian
- **Parameters**
  - `buffer` - the raw data buffer
  - `off` - the offset into the buffer where the long resides
  - `l` - A number of type long

`public final void` **writeLELong**`( long `**l**` )`

- **Usage**
  - Writes a number of type long in little endian
- **Parameters**
  - `l` - A number of type long

`public final void` **writeLEShort**`( short `**s**` )`

- **Usage**
  - Writes a number of type short in little endian
- **Parameters**
  - `s` - A number of type short

`public final void` **writePoint**`( byte [] `**b**`, int `**off**`, Point `**point**` )`

- **Usage**
  - Writes the given point to the given buffer at the given location. The point is written as a double representing x followed by a double representing y.

- **Parameters**
  - `b` - the data buffer
  - `off` - the offset into the buffer where writing should occur
  - `point` - the point to write

`public final void` **writePoint**`( Point` **point** `)`

- **Usage**
  - Writes the given point to the file as a double representing x followed by a double representing y.
- **Parameters**
  - `point` - the point to write

`public final void` **writeSpatialIndexRecord**`( IndexEntry` **record** `)`

- **Usage**
  - Writes the specified spatial index record to the file.

    Used to write records to flat file (.shx) spatial index's
- **Parameters**
  - `record` - the record to write out

`public final void` **writeString**`( String` **string**`, int` **length** `)`

- **Usage**
  - Writes a string of specified length to the file. If the specified string is shorter it is zero filled. If it is longer it is trimmed to fit.
- **Parameters**
  - `string` - the string to write to the file
  - `length` - the length of the string

Class **ShapeRecord**

A wrapper for Shape objects that holds all the associated information needed by the IO system.

extends java.lang.Object

FIELDS

- public IndexEntry index
  - Record number, offset, content length and type
- public Shape shape
  - The Geometry this record refers to
- public int partOffsets
  - Offsets to sub parts of the shape
- public int numPoints
  - Total number of points in shape
- public boolean clipped
  - indicates whether this shape has been clipped

CONSTRUCTORS

`public` **ShapeRecord**`( IndexEntry` **indexRecord** `)`

- **Usage**
  - Constructor. Adds the information already retrieved from the index file for this shape to its record. We assume that the information in the index matches exactly whats in the actual record so that information is not retieved again. No check of consistency is done.
- **Parameters**
  - `indexRecord` - the corresponding records index data

Methods

`public Object` **getAttribute**`( String` **key** `)`

- **Usage**
  - Gets the attribute value as specified by its name
- **Parameters**
  - `key` - the attribute name
- **Returns** - the attribute value (String or Double)

`public void` **setAttributes**`( String [] ` **keys**`, Object []` **values** `)`

- **Usage**
  - Add attributes to this shapes record. Attributes are held in the shape record instead of the shape itself because they are used by the query engine to decide which shapes need to be read from the file.

    Note: Multiple calls to this method cause a replacing of the attributes not an addition of attributes.
- **Parameters**
  - `keys` - the attribute names
  - `values` - the attribute values

Class **SHPFile**

The Main Shape file containing the shape records.

The main file (.shp) contains a fixed-length file header followed by variable-length records. Each variable-length record is made up of a fixed-length record header followed by variable-length record contents.

extends ie.tcd.cs.dsg.hermes.gis.io.ESRIFile

Constructors

`public` **SHPFile**`( File` **f** `)`

- **Usage**
  - Opens the file and reads the shapefile header
- **Parameters**
  - `f` - the file to access

METHODS

`public final void` **appendGeometry**`( ShapeList` **shapeRecords** `)`

- **Usage**
  - Adds new geometry to the end of the file. The new offsets and contentlengths are written back to the shape records and the file header is updated with the new file length and bounds.
- **Parameters**
  - `shapeRecords` - shape records containing geometry to write to file

`public final ShapeList` **readGeometry**`( ShapeList` **records** `)`

- **Usage**
  - Reads geometry from a .shp file
- **Parameters**
  - `records` - The index data retreived from the .shx file
- **Returns** - ShapeList A list of geometry

`public final ShapeList` **readPointGeometry**`( ShapeList` **records** `)`

- **Usage**
  - Iterates through the given input stream to contruct geometry objects
- **Parameters**
  - `records` - A list of offsets obtained by iterating through the associated SHX file
- **Returns** - list A `ShapeList` that contains the collection of objects created by iterating through this input stream

`public final ShapeList` **readPolyGeometry**`( ShapeList` **records**`, int` **shapeType** `)`

- **Usage**
  - Iterates through the given input stream to construct geometry objects
- **Parameters**
  - `shapeType` - the type of shape to read
  - `records` - A list of offsets obtained by iterating through the associated SHX file
- **Returns** - list A `ShapeList` that contains the collection of objects created by iterating through this input stream

`public final void` **writeGeometry**`( ShapeList` **indexData** `)`

- **Usage**
  - Writes geometry to a .shp file, overwriting any data currently in the file.
- **Parameters**
  - `indexData` - The index data retreived from the .shx file

`public final void` **writePointGeometry**`( ShapeList` **shapeRecords** `)`

- **Usage**
  - Writes point geometry to the class scope LittleEndianOutputStream.
- **Parameters**
  - `shapeRecords` - the ShapeRecords to write to the file

`public final void` **writePolyGeometry**`( ShapeList` **shapeRecords** `)`

- **Usage**
  - Writes polygon geometry to the class scope `LittleEndianInputStream`.
- **Parameters**
  - `shapeRecords` - The list of geometry objects to save

CLASS **SHPFileReport**

Reports on the contents of the SHP file specified.

extends java.lang.Object

CONSTRUCTORS

`public` **SHPFileReport**`( File` **f** `)`

METHODS

`public static void` **main**`( String [] ` **args** `)`

- **Usage**
  - Main method
- **Parameters**
  - `args` -

CLASS **SHXFile**

A class representing a shape index file. Reads index data from a .shx file.

The index file (.shx) contains a 100-byte header followed by 8-byte, fixed-length records.

extends ie.tcd.cs.dsg.hermes.gis.io.ESRIFile

implements ie.tcd.cs.dsg.hermes.gis.index.ShapeIndex

CONSTRUCTORS

`public` **SHXFile**`( File` **f** `)`

- **Usage**
  - Chains an input stream with a Little EndianInputStream

`public` **SHXFile**`( File` **f**`, SHPFile` **shp** `)`

Methods

`public boolean` **containsRecord**`( int` **recordNumber** `)`

`public IndexEntry` **getIndex**`( )`

- **Usage**
  - Gets the index data from the file.
- **Returns** - the index from disk or cache.

`public IndexEntry` **getRecord**`( int` **recordNumber** `)`

- **Usage**
  - Gets the index record for a specified record number. Assumes that records appear in the index in sequential order and no records are skipped.
- **Parameters**
  - `recordNumber` -
- **Returns** - the record number or null if the record number does not exist int he index

`public void` **getRecordOffsets**`( ShapeList` **list** `)`

`public void` **insert**`( IndexEntry` **entry** `)`

`public void` **insert**`( ShapeList` **records** `)`

`public boolean` **isEmpty**`( )`

Class **SSXFile**

A Spatial Index is a variation on a Shape Index, adding the bounding box of the shape to the index.
When reading the Shape file, the content length is the length of the record's contents, exclusive of the record header (8 bytes). So the size that we need to read in from the Shape file is actually denoted as ((contentLength * 2) + 8). This converts from 16bit units to 8 bit bytes and adds the 8 bytes for the record header.

extends ie.tcd.cs.dsg.hermes.gis.io.ESRIFile

implements ie.tcd.cs.dsg.hermes.gis.index.spatial.SpatialIndex

Fields

- public static final String PROPERTY_VALUE
  - The value of the SPATIAL_INDEX_PROPERTY if this class is to be used

Constructors

`public` **SSXFile**`( File` **f** `)`

- **Usage**
  - Opens a pre-generated spatial index for reading. If the spatial index is reasonably small it is read entirely into memory otherwise it is accessed from disk on each query.
- **Parameters**
  - `f` - a ssx fiel to open

`public` **SSXFile**`( File` **f**`, SHPFile` **shp** `)`

- **Usage**
  - Constructs a new Spatial index based on the data in the shp file specified.
- **Parameters**
  - `f` - the file to write the spatial index to

Methods

`public void` **delete**`( IndexEntry` **entry** `)`

`public void` **delete**`( Rectangle` **mbr** `)`

`public IndexEntry` **getIndex**`( )`

- **Usage**
  - Gets the index data from the file.
- **Returns** - the index from disk or cache.

`public String` **getName**`( )`

`public void` **getRecordOffsets**`( ShapeList` **list** `)`

`public void` **insert**`( IndexEntry` **entry** `)`

`public void` **insert**`( ShapeList` **records** `)`

`public boolean` **isEmpty**`( )`

`public ShapeList` **search**`( Rectangle` **query** `)`

- **Usage**
  - Locates records in the shape file that intersect with the given rectangle. The spatial index is searched for intersections and the appropriate records are read from the shape file.
- **Returns** - an array of offsets

Class **TestDataSource**

extends ie.tcd.cs.dsg.hermes.gis.io.AbstractShapeDataSource

Constructors

`public` **TestDataSource**`( )`

Methods

`public void` **close**`( )`

`public ShapeList` **getAttributesForRecords**`( ShapeList` **records** `)`

`public int` **getBytesRead**`( )`

`public int` **getBytesWritten**`( )`

`public Rectangle` **getExtents**`( )`

`public ShapeList` **getGeometryForRecords**(`ShapeList` **records**)

`public SpatialIndex` **getIndex**( )

`public String` **getName**( )

`public ShapeList` **getRecords**(`Rectangle` **area**)

`public ScaleRange` **getScale**( )

`public int` **getShapeType**( )

`public boolean` **hasGeometry**( )

D.3 Package ie.tcd.cs.dsg.hermes.gis.io.multiscale

Package Contents *Page*

Classes

Class **MultiScaleShapeDataSource**

Interface to data sources supporting multiple resolutions (scales)

extends ie.tcd.cs.dsg.hermes.gis.io.AbstractShapeDataSource

Constructors

`public` **MultiScaleShapeDataSource**`( )`

Methods

`public abstract void` **addGeometry**`( ShapeList` **records**`, Rectangle` **extent**`, ScaleRange` **scale** `)`

- **Usage**
  - Adds new geometry to the data source and updates index's. It is assumed that the Shapes share the same coordinate system as the existing Shapes stored in the datasource and are the same type of Shape.

    Note. Single shapes or multipart shapes only. i.e. adding multiple points at once will cause an exception.
- **Parameters**

  - `records` - the Shapes to add
  - `extent` - The MBR of the data to be added
  - `scale` - the scale at which the specified geometry is viewable

`public abstract ShapeList` **getGeometryForRecords**`( ShapeList` **records**`, float` **scale** `)`

- **Usage**
  - Gets the most appropriate versions of the specified geometry for the specified scale. This will result in some shapes being removed from the list and others being generalised before being returned.
- **Parameters**
  - `records` - the records that need geometry retrieved
  - `scale` - the scale at which the resulting geometry will be rendered
- **Returns** - the same shapelist with the geometry added

`public abstract ShapeList` **getRecordCache**`( float` **scale** `)`

- **Usage**
  - Returns the cache of records belonging to this data source
- **Parameters**
  - `scale` - the scale at which the specified geometry is viewable
- **Returns** - the cache of shape records

`public abstract void` **setRecordCache**`( ShapeList` **list**`, float` **scale** `)`

- **Usage**
  - Overwrites the current shape cache with this one.
- **Parameters**
  - `scale` - the scale at which the specified geometry is viewable
  - `list` - cached Shape objects

CLASS **MultiScaleShapeFile**

Multi-resolution, data store for scale independent spatial data. It is possible to read data in any resolution from this data source. As the scale increases the level of detail and generalisation reduces the dimensionality and information density of the returned geometry. Extensive use is made of caching to reduce the need to reproduce map simplification calculations. The version of a feature at the lowest scale (i.e. most detailed) is considered the authoritive definition of that feature and is the one that will be exchanged with peers.

extends ie.tcd.cs.dsg.hermes.gis.io.multiscale.MultiScaleShapeDataSource

FIELDS

- public static Comparator ascendingScaleComparator
  - Sorts files alphabetically. Public for unit testing

CONSTRUCTORS

`public` **MultiScaleShapeFile**`( ESRIShapeFile` **master** `)`

- **Usage**
  - Opens multiple shapefiles representing the same features at different resolutions and levels of detail. The files share file name components with the original main shape file.

    This constructor opens a multiscale datasource from a plain single scale shape file by opening any additional files necessary. The specified shapefile must be the master file.
- **Parameters**
  - `master` - The master shape file (lowest scale)

METHODS

`public void` **addGeometry**`(` `ShapeList` **shapeRecords**, `Rectangle` **extent** `)`

- **Usage**
  - Method should not be called. If it is it will place the geometry in the master shapefile.
- **See Also**
  - `SpatialDataSource.addGeometry(ShapeList, Rectangle)`

`public void` **addGeometry**`(` `ShapeList` **records**, `Rectangle` **extent**, `ScaleRange` **scale** `)`

`public void` **clearRecordCache**`( )`

`public ShapeList` **getAttributesForRecords**`(` `ShapeList` **records** `)`

`public Rectangle` **getExtents**`( )`

- **Usage**
  - Gets the extents for the data source. If there is no data in the datasource then empty extents with no area will be returned. Otherwise the extents of the master file are returned which should have been read from the files header and maintained during updates.
- **See Also**
  - `SpatialDataSource.getExtents()`

`public ShapeList` **getGeometryForRecords**`(` `ShapeList` **records** `)`

`public ShapeList` **getGeometryForRecords**`(` `ShapeList` **records**, `float` **scale** `)`

`public SpatialIndex` **getIndex**`( )`

`public String` **getName**`( )`

- **Usage**
  - Returns the name for the master shapefile. Any multi-resolution caches will not affect the name. The toString() method can be used to see what caches have been created and their status.

- **See Also**
  - `SpatialDataSource.getName()`

`public ShapeList` **getRecordCache**`( )`

`public ShapeList` **getRecordCache**`( float` **scale** `)`

`public ShapeList` **getRecords**`( Rectangle` **area** `)`

`public ScaleRange` **getScale**`( )`

- **Usage**
  - There is no scale for a multi-resolution data source. Anywhere between 0 and infinity is OK. This method will always return 0.
- **See Also**
  - `SpatialDataSource.getScale()`

`public int` **getShapeType**`( )`

`public boolean` **hasGeometry**`( )`

`public void` **removeGeometry**`( ShapeRecord` **record** `)`

`public void` **setRecordCache**`( ShapeList` **list** `)`

`public void` **setRecordCache**`( ShapeList` **list**`, float` **scale** `)`

`public void` **setScale**`( ScaleRange` **scale** `)`

- **Usage**
  - This method should never really be called because it dosen't make sense for a multi-resolution data source. There is not maximum scale!
- **See Also**
  - `SpatialDataSource.setScale(ie.tcd.cs.dsg.hermes.gis.io.multiscale.ScaleR`

Class ScaleRange

The ScaleRange class manages the dividing of scales into suitable ranges for caching. This is used so that a cache can be created that caters for a range of scales. This means that a cache is not needed for every scale point.
The size of the scale ranges is determined by a factor from the properties file that defines the logarithmically increasing scales.

extends java.lang.Object

Fields

- public static final String SCALE_RANGE_PROPERTY
  - The properties file key for locating the scale multiplier
- public static final float MIN_SCALE
  - The smallest scale multiplier

Methods

`public int` **compareTo**`( ScaleRange` **sr** `)`

- **Usage**
  - Compare the size of scale ranges.
- **Parameters**
  - `sr` -
- **Returns** - the value 0 if the ScaleRange string is equal to this string; a value less than 0 if this ScaleRange is less than the ScaleRange argument; and a value greater than 0 if this ScaleRange is greater than the ScaleRange argument.

`public static ScaleRange` **getRange**`( float` **scale** `)`

- **Usage**
  - Gets the range that this scale falls between.
- **Parameters**

  – `scale` - the scale

- **Returns** - the range that this scale is in

`public ScaleRange` **getRangeAbove**`( )`

- **Usage**
  – Returns the next highest scale range. A new range is constructed if necessary.
- **Returns** - the next highest scale range

`public ScaleRange` **getRangeBelow**`( )`

- **Usage**
  – Gets the next lowest scale range or null if this is the minimum scale
- **Returns** - the next lowest scale range or null if this is the minimum scale

`public boolean` **isWithinRange**`( float  s )`

- **Usage**
  – Tests to see if the specified scale is within the bounds of this range
- **Parameters**
  – `s` - the scale
- **Returns** - true if is between the min and max values

D.4 Package ie.tcd.cs.dsg.hermes.gis.event

Package Contents *Page*

Interfaces

INTERFACE **InputModeListener**

Class's implementing this interface respond to changes in keypad input mode.

implements java.util.EventListener

METHODS

`public void` **inputModeChanged**`( InputModeEvent e )`

- **Usage**

  – The keypad input mode has changed in the KeyHandler class. The event contains the mode that is currently in use.

- **Parameters**
  – `e` - an input mode event

Interface **LayerListener**

Interface for listening to LayerEvents. LayerEvent is fired when something fundamental about the Map layers changes (e.g. layer added, removed or made visible).

implements java.util.EventListener

Methods

`public void` **layerChanged**`( LayerEvent` **e** `)`

- **Usage**
  – Invoked when there has been a change to the Map layers.
- **Parameters**
  – `e` - LayerEvent

Interface **MapMovementListener**

Listens for requests to recenter the map.

implements java.util.EventListener

Methods

`public void` **center**`( CenterEvent` **evt** `)`

- **Usage**
  – Center the map
- **Parameters**

  – `evt` - an event specifying the coordinates to center the map on

`public void` **pan**`( PanEvent` **evt** `)`

- **Usage**
  – Pan the map in a specified direction
- **Parameters**
  – `evt` - a pan event specifying the direction and amount to pan the map

`public void` **rotate**`( RotateEvent` **re** `)`

- **Usage**
  – Rotate the map about its center
- **Parameters**
  – `re` - the distance in degrees to rotate the map

`public void` **zoom**`( ZoomEvent` **evt** `)`

- **Usage**
  – Zoom the map
- **Parameters**
  – `evt` - an event indicating the amount to zoom

INTERFACE **ProjectionListener**

Interface for listening to ProjectionEvents. ProjectionEvent is fired when something fundamental about the Map changes (e.g. when width, height, scale, type, center, etc changes).

implements java.util.EventListener

Methods

`public void` **projectionChanged**`( ProjectionEvent e )`

- **Usage**
  - Invoked when there has been a fundamental change to the Map. Layers are expected to recompute their graphics (if this makes sense), and then `repaint()` themselves.
- **Parameters**
  - `e` - ProjectionEvent

Classes

Class **CenterEvent**

An event to request the map should recenter to a new latitude and longitude.

extends java.util.EventObject

Constructors

`public` **CenterEvent**`( Object source, int x, int y )`

- **Usage**
  - Construct a CenterEvent.
- **Parameters**
  - `source` - the source bean
  - `x` - screen point to center on
  - `y` - screen point to center on

Methods

`public int` **getX**`( )`

- **Returns** - Returns the _x.

`public int` **getY**`( )`

- **Returns** - Returns the _y.

Class **InputModeEvent**

An event indicating a change in the input mode (i.e. function of the keypad). These events are used to indicate to the user which mode of operation the keypad is in by displaying an icon.

extends java.util.EventObject

Constructors

`public` **InputModeEvent**`( Object` **source**`, int` **inputMode** `)`

- **Parameters**
  - `source` - event source (an AWT container)
  - `inputMode` - See KeyHandler for possible values

Methods

`public int` **getInputMode**`( )`

- **Usage**
  - Gets the input mode
- **Returns** - the input mode

Class **LayerEvent**

An event indicating that a layer has changed and should be repainted.

extends java.util.EventObject

Constructors

`public` **LayerEvent**`(` `MapLayer` **layer** `)`

- **Usage**
  - Construct a LayerEvent.
- **Parameters**
  - `layer` - MapLayer that is source of event

`public` **LayerEvent**`(` `MapLayer` **layer**, `boolean` **zOrderChanged** `)`

- **Usage**
  - Construct a LayerEvent using this value for the layers Z-order
- **Parameters**
  - `layer` - MapLayer that is source of event
  - `zOrderChanged` - true if a layers z-order has changed

Methods

`public MapLayer` **getLayer**`( )`

- **Usage**
  - Gets the reference to the layer that changed
- **Returns** - the map layer that caused the event

`public boolean` **zOrderChanged**`( )`

- **Usage**
  - Tests to see if the Z-order of the layers has changed.
- **Returns** - true if the layers Z-order has changed.

CLASS **PanEvent**

An event to request the map to pan. Event designates the direction and magnitude (relative to map dimensions) to pan the map.

extends java.util.EventObject

FIELDS

- public static final int NORTH
- public static final int NORTH_EAST
- public static final int EAST
- public static final int SOUTH_EAST
- public static final int SOUTH
- public static final int SOUTH_WEST
- public static final int WEST
- public static final int NORTH_WEST

CONSTRUCTORS

`public` **PanEvent**`( Object` **source**`, int` **direction**`, int` **arcDistance** `)`

- **Usage**
  - Create a PanEvent with source Object and direction.

- **Parameters**
  - `source` - Object
  - `direction` - N, NE, E, SE, S, SW, W, NW
  - `arcDistance` - 0.0 <= x <= 1.0

Methods

`public float` **getAzimuth**`( )`

- **Usage**
  - Get azimuth. Azimuth is the horizontal component of a direction (compass direction), measured around the horizon, from the north toward the east.
- **Returns** - float decimal degrees

`public int` **getDirection**`( )`

- **Usage**
  - Get the direction of pan.
- **Returns** - int direction

`public float` **getPanPercent**`( )`

- **Usage**
  - Used to calculate the distance to move the map center based on the percentage of screen width to move.
- **Returns** - distance in percentage of visible area

Class **PDAKeyHandler**

Used to handle KeyEvents coming from a PDA keypad. Specifically the HP IPAQ hx2400 four button keypad

extends java.awt.event.KeyAdapter

Fields

- public static final int MODE_ZOOM_ROTATE
  - In this mode arrow keys zoom or rotate the map

- public static final int MODE_PAN
  - In this mode arrow keys pan the map

METHODS

`public void` **addInputModeListener**`( InputModeListener` **listener** `)`

- **Usage**
  - Adds an input mode listener to this class. Only a single listener may be added. Further calls to this method will overwrite the listener.
- **Parameters**
  - `listener` - an `InputModeListener` instance.

`public static PDAKeyHandler` **getInstance**`( MapContext` **map** `)`

- **Usage**
  - Singleton pattern alternative to constructor
- **Parameters**
  - `map` - the map that will be moved
- **Returns** - an instance of this class

`public static int` **getMode**`( )`

- **Usage**
  - Gets the current input mode.
- **Returns** - the current input mode

`public void` **keyPressed**`( KeyEvent` **e** `)`

- **Usage**
  - Called when a key is pressed. Press events that arrive within 50ms of each other are treated as a single press (second key ignored).
- **Parameters**

  – `e` - a `KeyEvent` instance

`public void` **setRotate**`( boolean` **rotate** `)`

- **Usage**
  – Modifies the keypad behaviour
- **Parameters**
  – `rotate` - allow the buttons to rotate the map

`public void` **setZoomRelative**`( boolean` **relative** `)`

- **Usage**
  – Modifies the keypad behaviour
- **Parameters**
  – `relative` - allow buttons to make relative changes to the zoom level

CLASS **ProjectionEvent**

An event indicating an updated projection.

extends java.util.EventObject

CONSTRUCTORS

`public` **ProjectionEvent**`( Object` **source**`, Projection` **aProj** `)`

- **Usage**
  – Construct a ProjectionEvent.
- **Parameters**
  – `source` - the object that raised the event
  – `aProj` - the `Projection` that changed

Methods

`public RotatableProjection` **getProjection**`( )`

- **Usage**
  - Get the Projection.
- **Returns** - Projection

Class **RotateEvent**

Rotate map event. Causes the map to rotate by a specified amount in a particular direction or to a specified point.

extends java.util.EventObject

Fields

- public static final transient int CLOCKWISE
- public static final transient int COUNTER_CLOCKWISE

Constructors

`public` **RotateEvent**`( Object` **source**`, int` **degrees** `)`

- **Parameters**
  - `source` - the event source
  - `degrees` - the distance to rotate

`public` **RotateEvent**`( Object` **source**`, int` **degrees**`, int` **direction** `)`

- **Parameters**
  - `source` - the event source
  - `degrees` - the distance to rotate
  - `direction` - the direction to rotate

Methods

`public int` **getDegrees**`( )`

- **Usage**
  - The degrees to rotate
- **Returns** - The degrees to rotate

`public int` **getDirection**`( )`

- **Usage**
  - Gets the direction the event is requesting a rotation
- **Returns** - the direction to rotate

Class **ZoomEvent**

An event to request that the map zoom in or out. Event specifies the type and amount of zoom of the map.

extends java.util.EventObject

Fields

- public static final transient int RELATIVE
  - Type that specifies that the amount should be used as a multiplier to the current scale.
- public static final transient int ABSOLUTE
  - Type that specifies that the amount should be used as the new scale.

Constructors

`public` **ZoomEvent**`( Object` **source**`, int` **type**`, float` **amount** `)`

- **Usage**
  - Construct a ZoomEvent.
- **Parameters**
  - `source` - the creator of the ZoomEvent.
  - `type` - the type of the event, refering to how to use the amount.
  - `amount` - the value of the ZoomEvent.

Methods

`public float` **getAmount**`( )`

- **Usage**
  - Get the amount of zoom.
- **Returns** - float

`public boolean` **isAbsolute**`( )`

- **Usage**
  - Check if the type is ABSOLUTE.
- **Returns** - boolean

`public boolean` **isRelative**`( )`

- **Usage**
  - Check if the type is RELATIVE.
- **Returns** - boolean

D.5 Package ie.tcd.cs.dsg.hermes.gis

Package Contents *Page*

Classes

Class **MapContext**

MapContext is the main component of MobileGIS. It manages and displays a map. A map is comprised of a projection and a list of layers, and this class has methods that allow you to control the projection parameters and to add and remove layers. Layers that are part of the map receive dynamic notifications of changes to the underlying view and projection.

extends java.lang.Object
implements ie.tcd.cs.dsg.hermes.gis.event.MapMovementListener, ie.tcd.cs.dsg.hermes.gis.event.LayerListener

Fields

- public static final Class projectionConstructorArgs
  - The constructor arguments for projections

Constructors

`public` **MapContext(** `Point` **center**, `float` **s**, `int` **w**, `int` **h**, `String` **projClass** `)`

- **Usage**
  - Private Constructor. Use static getInstance()

- **Parameters**
  - `center` - The center of the map
  - `s` - the scale of the map
  - `w` - the width of the screen
  - `h` - the height of the screen
  - `projClass` - the projection to use

Methods

`public void` **addLayer**`( MapLayer` **layer** `)`

- **Usage**
  - Adds the specified layer to the map
- **Parameters**
  - `layer` - the MapLayer to add

`public void` **center**`( CenterEvent` **evt** `)`

`public BufferedMapLayer` **getBufferedMapLayers**`( )`

- **Usage**
  - Gets just the Buffered map layers
- **Returns** - an array of the buffered may layers

`public DynamicMapLayer` **getDynamicMapLayers**`( )`

- **Usage**
  - Gets just the dynamic map layers
- **Returns** - an array of DynamicMapLayers

`public MapLayer` **getLayer**`( String` **layerName** `)`

- **Usage**
  - Gets a layer specified by name.
- **Parameters**

  - `layerName` - the String identifier for a particular layer.

- **Returns** - a maplayer or null if one is not found

`public LinkedList` **getLayers**`( )`

- **Usage**
  - Gets all the layers as an array.
- **Returns** - all the map layers as an array.

`public int` **getNumberLayers**`( )`

- **Usage**
  - returns the number of layers currently loaded (visible or not)
- **Returns** - the number of layers currently loaded

`public RotatableProjection` **getProjection**`( )`

- **Usage**
  - Returns the current projection. Used for initialisation purposes only as all further references to projections are delivered via events.
- **Returns** - the current projection

`public Dimension` **getScreenSize**`( )`

- **Returns** - Returns the screenSize.

`public void` **pan**`( PanEvent` **evt** `)`

`public void` **removeLayer**`( MapLayer` **layer** `)`

- **Usage**
  - Removes the specified layer
- **Parameters**
  - `layer` - the Map Layer to remove

`public void` **removeLayer**`( String` **layerName** `)`

- **Usage**
  - Remove a layer specified by its name

`public void` **resize**`( int` **width**`, int` **height** `)`

- **Usage**
  - ComponentListener interface method. Should not be called directly. Invoked when component has been resized, and kicks off a projection change.

`public void` **rotate**`( RotateEvent` **evt** `)`

`public void` **zoom**`( ZoomEvent` **evt** `)`

Class **MobileGIS**

Creates map based interfaces for mobile devices from ESRI Shapefile GIS datasets.

extends java.awt.Frame
implements ie.tcd.cs.dsg.hermes.gis.tools.benchmark.ClientStateMonitor

Fields

- public static final String WORKING_DIRECTORY_PROPERTY
  - Property for working directory
- public static final String DEFAULT_LAYERSET_PROPERTY
  - Tells the component to load the default set of layers
- public static boolean DEBUG
  - Print debugging information to stdout?
- public static Dimension screenSize
  - The screen area in pixels

- public static String workingDirectory
  - Current working directory [Passed as parameter on execution]
- public static Logger log
  - Instance of component for logging

CONSTRUCTORS

`public` **MobileGIS**`( )`

METHODS

`public static MapCanvas` **getMapCanvas**`( )`

- **Usage**
  - Provides class extending GIS access to the map
- **Returns** - the map canvas

`public static MapContext` **getMapContext**`( )`

- **Usage**
  - Gets the MapContext created from the properties file durining initialisation.
- **Returns** - the current map context

`public static Properties` **getProperties**`( )`

- **Usage**
  - Gets the properties fiel for this application. If the working directory has not been passed as a paremeter and set then use the systems current working directory.
- **Returns** - The packages properties

`public static void` **init**`( )`

- **Usage**

  – Framework initialisation method. To use the map canvas as a GUI component of other applications call the init() method and then retrieve the MapCanvas and MapContext statically from this class.

D.6 Package ie.tcd.cs.dsg.hermes.gis.geometry

Package Contents *Page*

Interfaces

INTERFACE Geometry

Root interface that is implemented by all Geometry classes

METHODS

`public boolean` **contains**`( float` **lat**`, float` **lon** `)`

- **Usage**
  - Tests whether or not the specified point is inside this polygon.
- **Parameters**

  - `lat` - the X coordinate of the point to test
  - `lon` - the Y coordinate of the point to test

- **Returns** - true if the point is inside this polygon

`public float` **getArea**`( )`

- **Usage**
  - Calculates and returns the area of the specified geometry. For Polygons, this is the total area inside the external ring less the total of any contained by interior rings. GeometryCollections (including MultiPolygons) are iterated through so the result is the sum of all polygons anywhere within the collection. Any geometry other than Polgyon or a collection returns 0;
- **Returns** - The total area of the Geometry.

`public Rectangle` **getBounds**`( )`

- **Usage**
  - Returns the bounding box of this polygon. This is the smallest rectangle with sides parallel to the X axis that will contain this polygon.
- **Returns** - the bounding box for this polygon

`public Point` **getCenter**`( )`

- **Usage**
  - Finds the centroid of the input geometry if input = point, line, polygon return a point that represents the centroid of that geom if input = geometry collection, return a multipoint that represents the centoid of each sub-geom
- **Returns** - a double array x, y

`public int` **getNumberPoints**`( )`

- **Usage**
  - Returns the number of points the shape is made up of
- **Returns** - the number of points in the shape

`public float` **`getPoints`**`( )`

- **Usage**
  - Gets the source points of the geometry.
- **Returns** - the source points of the geometry.

INTERFACE **ShapeListFilter**

Classes implementing this interface are used to remove records from a shape list.

METHODS

`public boolean` **`accept`**`( ShapeRecord` **`record`** `)`

- **Usage**
  - Tests the specified record to see if it should remain or be filtered out.
- **Parameters**
  - `record` - a ShapeRecord to test
- **Returns** - true if the record remain, false if it should be removed from the list

Classes

CLASS **Annotation**

Adds the ability to use any shape as a marker for drawing annotation or labels on the map

extends ie.tcd.cs.dsg.hermes.gis.geometry.Point

FIELDS

- public static final int DEFAULT_TEXT_ANGLE
- public static final int DEFAULT_TEXT_SIZE

Constructors

`public` **Annotation**`( Point` **p**`, String` **text** `)`

- **Parameters**
  - `p` - the point at which the annotation is anchored
  - `text` - a string that should be displayed

`public` **Annotation**`( Point` **p**`, String` **text**`, int` **textSize**`, int` **textAngle** `)`

- **Parameters**
  - `p` - the shape indicating where the annotation should be located

Methods

`public void` **render**`( Graphics` **g**`, RotatableProjection` **projection**`, Style` **style** `)`

Class **Line**

A line is a polyline with just a start and end point i.e. a segment or edge.

extends ie.tcd.cs.dsg.hermes.gis.geometry.Polyline

Constructors

`public` **Line**`( float []` **points** `)`

- **Usage**
  - Constructor with two points.
- **Parameters**
  - `points` -

`public` **Line**`( Point` **p1**`, Point` **p2** `)`

- **Usage**
  - Constructs a line between the two specified points
- **Parameters**
  - `p1` - the first point
  - `p2` - the second point

METHODS

`public boolean` **intersects(** `Line` **l2** `)`

- **Usage**
  - Tests if the line segment from (X1, Y1) to (X2, Y2) intersects this line segment.
- **Parameters**
  - `l2` - the coordinates of the beginning of the specified line segment
- **Returns** - if this line segment and the specified line segment intersect each other; `false` otherwise.

`public boolean` **intersects(** `Polygon` **p** `)`

- **Usage**
  - Tests is this line intersects the specified polygon
- **Parameters**
  - `p` - a polygon to check for intersection
- **Returns** - true if this line intersects the soecified polygon

`public boolean` **intersects(** `Shape` **s** `)`

- **Usage**
  - Tests if this lien intersects the specified shape. TODO: Add this method to unit testing for line intersection
- **Parameters**

  - `s` - a shape to test

- **Returns** - true if there is an intersection

`public void` **render**`( Graphics` **g**`, RotatableProjection` **projection**`, Style` **style** `)`

Class **Point**

Encapsulates latitude and longitude coordinates in decimal degrees.

extends ie.tcd.cs.dsg.hermes.gis.geometry.Shape
implements java.lang.Cloneable

Fields

- public static final int DEFAULT_RADIUS
  - Default radius in Pixels
- public static final float EQUIVALENT_TOLERANCE
- public float lat
  - Latitude of point, decimal degrees.
- public float lon
  - Longitude of point, decimal degrees.

Constructors

`public` **Point**`( double` **lon**`, double` **lat** `)`

- **Usage**
  - Construct a LatLonPoint from raw ESRI double lat/lon.
- **Parameters**
  - `lat` - latitude in decimal degrees

  – `lon` - longitude in decimal degrees

`public` **Point**`( float` **lat**`, float` **lon** `)`

- **Usage**
  – Construct a LatLonPoint from raw float lat/lon in decimal degrees.
- **Parameters**
  – `lat` - latitude in decimal degrees
  – `lon` - longitude in decimal degrees

`public` **Point**`( Point` **pt** `)`

- **Usage**
  – Copy construct a LatLonPoint.
- **Parameters**
  – `pt` - LatLonPoint

Methods

`public float` **azimuth**`( Point` **toPoint** `)`

- **Usage**
  – Find the azimuth to another point, based on the sphercal earth model.
- **Parameters**
  – `toPoint` - LatLonPoint
- **Returns** - the azimuth 'Az' east of north from this point bearing toward the one provided as an argument.(-PI <= Az <= PI).

`public float` **distance**`( Point` **toPoint** `)`

- **Usage**
  – Find the distance to another LatLonPoint, based on a earth spherical model.
- **Parameters**

  - `toPoint` - LatLonPoint

- **Returns** - distance, in meters.

`public float` **getLatitude**`( )`

- **Usage**
  - Get normalised latitude.
- **Returns** - float latitude in decimal degrees

`public float` **getLongitude**`( )`

- **Usage**
  - Get wrapped longitude.
- **Returns** - float longitude in decimal degrees

`public boolean` **intersects**`( double` **x**`, double` **y**`, double` **w**`, double` **h** `)`

`public final Point` **pointAtAzDist**`( float` **c**`, float` **Az** `)`

- **Usage**
  - Calculate point at azimuth and distance from another point. Returns a LatLonPoint at arc distance ‘c’ in direction ‘Az’ from start point.
- **Parameters**
  - `c` - distance in meters
  - `Az` - direction in degrees north
- **Returns** - Point

`public void` **render**`( Graphics` **g**`, RotatableProjection` **projection**`, Style` **style** `)`

`public void` **setLatitude**`( float` **lat** `)`

`public void` **setLatLon**`( float` **lat**`, float` **lon** `)`

- **Usage**
  - Set latitude and longitude.

- **Parameters**
  - `lat` - latitude in decimal degrees
  - `lon` - longitude in decimal degrees

`public void` **setLongitude**`( float` **lon** `)`

Class **Poly**

A Shape with multiple straight edges

extends ie.tcd.cs.dsg.hermes.gis.geometry.Shape

Constructors

`public` **Poly**`( )`

- **Usage**
  - Default constructor. Used for incrementally building new polygons during simplification.

`public` **Poly**`( int` **initialSize** `)`

- **Parameters**
  - `initialSize` - the initial number of points. Used to dimension an array.

Methods

`public void` **addVertex**`( float` **lat**`, float` **lon**`, boolean` **updateBounds**
`)`

- **Usage**
  - Adds a single point as a new vertex in the poly
- **Parameters**
  - `lat` - the longitude component of a `Point` to add as a new vertex
  - `lon` - the latitude component of a `Point` to add as a new vertex

`public void` **addVertex**`( Point` **p** `)`

- **Usage**
  - Overloads other addVertex method. Bounds will be updated to reflect change in points. Other method should be called instead. TODO deprecate this method. Force decision on whether bounds would be updated.
- **Parameters**
  - `p` - a `Point` to add as a new vertex

`public boolean` **contains**`( float` **lat**`, float` **lon** `)`

`public boolean` **contains**`( Shape` **s** `)`

- **Usage**
  - Check to see if the specified shape is contained by this polygon. This is achieved by checking each of the polygons vertices in turn. The bounding boxes are checked first as a simple quick reject.
- **Parameters**
  - `s` - the polygon to test
- **Returns** - true if the specified polygon is completely contained by this object

`public boolean` **equals**`( Object` **o** `)`

- **Usage**
  - Polys are equals when they have the same coordiantes.
- **See Also**
  - `java.lang.Object.equals(java.lang.Object)`

`public float` **getArea**`( )`

`public Point` **getCenter**`( )`

`public float` **getLatLonPoints**`( )`

- **Usage**

  - Used by simplification (Nth Point) to modify points

- **Returns** - the underlying coordinates

`public int` **getNumberPoints**`( )`

`public float` **getPoints**`( )`

- **Usage**
  - Used to access the underlying array of points. Used in graph building and IO code.
- **Returns** - the underlying float array

`public void` **setPoints**`( float []` **points** `)`

- **Usage**
  - Sets the points this Polygon represents
- **Parameters**
  - `points` - the latlonpoints array lat,lon,lat . .

Class **Polygon**

Represents a simple Polygon

extends ie.tcd.cs.dsg.hermes.gis.geometry.Poly

Constructors

`public` **Polygon**`( )`

`public` **Polygon**`( float []` **points** `)`

- **Parameters**
  - `points` - an array of long / lat points

Methods

`public void` **render**`( Graphics` **g**`, RotatableProjection` **projection**`, Style` **style** `)`

CLASS **Polyline**

Represents a polyline

extends ie.tcd.cs.dsg.hermes.gis.geometry.Poly

CONSTRUCTORS

`public` **Polyline**`( )`

- **Usage**
  - Constructs empty Polyline. AddVertex should be called next.

`public` **Polyline**`( float []` **points** `)`

METHODS

`public void` **render**`( Graphics` **g**`, RotatableProjection` **projection**`, Style` **style** `)`

CLASS **Rectangle**

MBR rectangle (Bounding Box)

extends ie.tcd.cs.dsg.hermes.gis.geometry.Polygon

FIELDS

- public static final Rectangle EMPTY
  - An empty bounding box (i.e. no area)
- public static final Rectangle PLANET
  - A bounding box covering the whole world

Constructors

`public` **Rectangle**`( )`

`public` **Rectangle**`( float [] `**points**` )`

- **Usage**
  - Constructor with two points.
- **Parameters**
  - `points` -

`public` **Rectangle**`( Point `**p**` )`

- **Parameters**
  - `p` -

`public` **Rectangle**`( Point `**min**`, Point `**max**` )`

- **Usage**
  - Constructs a rectangle enclosing the two specified points
- **Parameters**
  - `min` - the first point
  - `max` - the second point

Methods

`public void` **addVertex**`( float `**lat**`, float `**lon**` )`

`public void` **addVertex**`( Point `**p**` )`

`public final boolean` **contains**`( float `**lat**`, float `**lon**` )`

`public boolean` **contains**`( Shape `**s**` )`

`public boolean` **equals**`( Object `**obj**` )`

`public float` **getArea**`( )`

- **Usage**
  - Returns the area of the bounding box in meters squared

- **See Also**
  - `ie.tcd.cs.dsg.hermes.gis.geometry.Poly.getArea()`

`public Rectangle` **getBounds**`( )`

`public final Point` **getCenter**`( )`

- **Usage**
  - Used to find origin in rotating
- **Returns** - the center point

`public int` **getNumberPoints**`( )`

`public final boolean` **intersects**`( Rectangle` **r** `)`

- **Usage**
  - Tests to see if the specified box intersects (overlaps) this box This means the two Box's share at least one internal point.
- **Parameters**
  - `r` - the rectangle to test against
- **Returns** - true if the specified rectangle intersects this one

`public final void` **union**`( Rectangle` **r** `)`

- **Usage**
  - Union's the specified bounding box with this one
- **Parameters**
  - `r` - the rectangle to merge with this one

Class **Shape**

Represents a shape that can be rendered on the screen

extends java.lang.Object

implements Geometry, java.lang.Cloneable

Constructors

`public` **Shape**`( )`

Methods

`public Rectangle` **getBounds**`( )`

`public static final double` **getMinimumDistance**`( Shape` **fromShape**`, Shape` **toShape** `)`

- **Usage**
  - Minimum distance concept and equation illustrated here: http://www.cs.mcgill.ca/ fzamal/Project/concepts.htm Used for nearest neighbour operations
- **Parameters**
  - `toShape` -
- **Returns** - minimum distance to shape

`public boolean` **isVisible**`( )`

- **Returns** - Returns the visible.

`public abstract void` **render**`( Graphics` **g**`, RotatableProjection` **projection**`, Style` **style** `)`

- **Usage**
  - Render the shape to the specified graphics context
- **Parameters**
  - `g` - the graphics context
  - `projection` - the projection to use
  - `style` - the style containing drawing attributes

`public void` **setVisible**`( boolean` **visible** `)`

- **Parameters**
  - `visible` - The visible to set.

CLASS **ShapeList**

A lightweight vector of Shape records. Performs better than Javas in built lists.

extends java.lang.Object

implements java.lang.Cloneable

FIELDS

- public int count
  - The number of items in the vector
- public ShapeRecord data
  - The items in the vector

CONSTRUCTORS

`public` **ShapeList**`( int` **initialSize** `)`

- **Parameters**
  - `initialSize` - the approximate initial number of list items

`public` **ShapeList**`( int` **initialSize**`, boolean` **unique** `)`

- **Parameters**
  - `initialSize` - the approximate initial number of list items
  - `unique` - Each record number may only appear once in list

`public` **ShapeList**`( ShapeList` **sRecords** `)`

- **Usage**
  - Constructor. Creates a deep clone of the supplied object
- **Parameters**
  - `sRecords` - a ShapeList of Geometry that should be cloned

Methods

`public void` **addAll**`( ShapeList` **sr** `)`

- **Usage**
  - Appends the specified collection of shape records to the end of this list.
- **Parameters**
  - `sr` - An existing ShapeList from which Shapes will be copied

`public void` **addElement**`( ShapeRecord` **rec** `)`

- **Usage**
  - Adds the specified component to the end of this vector, increasing its size by one.
- **Parameters**
  - `rec` - a shape record object

`public final void` **clear**`( )`

- **Usage**
  - Clears the list of all data. The data is still referenced but count shows it as being empty.

`public Object` **clone**`( )`

`public ShapeList` **clone**`( boolean` **deep** `)`

- **Usage**
  - Partial Deep clone (IndexEntrys but not the ShapeRecord objects)
- **Parameters**
  - `deep` -
- **Returns** - a clone of this list

`public final boolean` **containsRecord**`( int` **recNo** `)`

- **Usage**

  – Tests if the specified record is in this list. Used when retrieving new geometry to avoid reading shapes from disk a second time.

    Note: Record numbers must be unique

- **Parameters**
  – `recNo` - the shape record number from the file
- **Returns** - true if the shape is loaded already

`public ShapeList` **filter**`( ShapeListFilter` **filter** `)`

- **Usage**
  – Filters this list returning a new list with the elements of this list that pass the filters test.
- **Parameters**
  – `filter` - a ShapeListFilter that tests the records in the list
- **Returns** - a new ShapeList with the elements that passed the test

`public final Object` **getAttribute**`( ShapeRecord` **record**`, String` **attribute** `)`

- **Usage**
  – See the general contract of the `getAttribute` method of `SpatialDataSource`.

    Note: Here for compatability but dosen't do anything. ShapeRecords should already have geometry
- **Parameters**
  – `record` -
  – `attribute` - the anem of the attribute
- **Returns** - the attribute object, Double or String

`public final Rectangle` **getExtents**`( )`

- **Usage**

  - See the general contract of the `getExtents` method of `SpatialDataSource`.

- **See Also**
  - `SpatialDataSource.getExtents()`

`public final ShapeRecord` **getRecord**`( int` **recNo** `)`

- **Usage**
  - Gets a record specified by its data source identifier and record number
- **Parameters**
  - `recNo` - the shape record number from the file
- **Returns** - the ShapeRecord if it exists or null

`public final ShapeList` **getRecords**`( Rectangle` **area** `)`

- **Usage**
  - See the general contract of the `getRecords` method of `SpatialDataSource`.

`public final int` **indexOf**`( ShapeRecord` **obj**`, int` **i** `)`

- **Usage**
  - Searches for the first occurrence of the given argument, beginning the search at index, and testing for equality using the equals method.
- **Parameters**
  - `obj` - an object
  - `i` - the non-negative index to start searching from.
- **Returns** - he index of the first occurrence of the object argument in this vector at position index or later in the vector

`public int` **insertElementAt**`( ShapeRecord` **obj**`, int` **i** `)`

- **Usage**

  – Inserts the specified object as a component in this vector at the specified index.

- **Parameters**
  – `obj` - an object
  – `i` - the index to insert at
- **Returns** - -1 if the element was not inserted, otherwise 0

`public final int` **lastIndexOf**`( ShapeRecord` **obj**`, int` **i** `)`

- **Usage**
  – Searches backwards for the specified object, starting from the specified index, and returns an index to it.
- **Parameters**
  – `obj` - an object
  – `i` - the index to start searching from.
- **Returns** - the index of the last occurrence of the specified object in this vector at position less than or equal to index in the vector,

`public final int` **removeElementAt**`( int` **i** `)`

- **Usage**
  – Deletes the component at the specified index. Each component in this vector with an index greater or equal to the specified index is shifted downward to have an index one smaller than the value it had previously. The size of this vector is decreased by 1.
- **Parameters**
  – `i` - the index of the object to remove.
- **Returns** - -1 if the element was not removed, otherwise 0

`public final void` **removeRecord**`( ShapeRecord` **record** `)`

- **Usage**

  – See the general contract of the `removeGeometry` method of `SpatialDataSource`.

- **See Also**
  –
  `SpatialDataSource.removeGeometry(ie.tcd.cs.dsg.hermes.gis.io.ShapeRecor`

`public final void` **replaceRecord**`( ShapeRecord` **record** `)`

- **Usage**
  – Replaces the record with the one specified it is is contained in the list.
- **Parameters**
  – `record` - the record to replace

`public final void` **setScale**`( float` **scale** `)`

- **Usage**
  – Sets the scale of all the records in the list to the specified scale range.
- **Parameters**
  – `scale` -

`public void` **sort**`( ShapeRecordComparator` **c** `)`

- **Usage**
  – Sorts the elements of the list based on the specified comparator.
- **Parameters**
  – `c` -

D.7 Package ie.tcd.cs.dsg.hermes.gis.ui

Package Contents *Page*

Interfaces

Interface **ScreenOverlay**

An interface for interface artifacts that the GIS rendering engine may need to overlay on a map layer.

All coordinates are in device screen coordinate space.

Methods

`public int` **getGeometry**`( )`

- **Usage**
  - Gets the outline of the arrow as arrays of x and y coordinate pairs for filling by the Graphics class.
- **Returns** - x and y coordinate arrays

`public void` **render**`( Graphics` **g**`, RotatableProjection` **projection**`, Style` **style** `)`

- **Usage**
  - Renders the screen overlay on the device screen

- **Parameters**
  - `g` - the graphics context
  - `projection` - the current map projection
  - `style` - the rendering style

Classes

Class **BufferedMapCanvas**

A Double buffered version of the map canvas. The original map canvas should never be used.

extends ie.tcd.cs.dsg.hermes.gis.ui.MapCanvas

implements ie.tcd.cs.dsg.hermes.gis.event.ProjectionListener

Constructors

`public` **BufferedMapCanvas**`( MapContext` **mapContext** `)`

- **Parameters**
  - `mapContext` - the map

Methods

`public void` **componentResized**`( ComponentEvent` **e** `)`

- **Usage**
  - Invoked when component has been resized. Layer buffer is nullified. and super.componentResized(e) is called.
- **Parameters**
  - `e` - ComponentEvent

`public void` **inputModeChanged**`( InputModeEvent` **e** `)`

`public void` **paint**`( Graphics` **g** `)`

`public void` **projectionChanged**`( ProjectionEvent` **e** `)`

`public void` **setBufferDirty**`( boolean` **value** `)`

- **Usage**
  - Marks the image buffer as dirty if value is false. On the next `paintChildren()`, we will call `paint()` on all Layer components.
- **Parameters**
  - `value` - boolean value indicating buffer state

CLASS **MapCanvas**

A graphical representation of a vector map .

extends java.awt.Canvas
implements java.awt.event.ComponentListener,

ie.tcd.cs.dsg.hermes.gis.event.LayerListener,

ie.tcd.cs.dsg.hermes.gis.event.InputModeListener

CONSTRUCTORS

`public` **MapCanvas**`( MapContext` **mapContext** `)`

- **Usage**
  - Constructor. Creates a new MapCanvas to show Map data from the Specified Index for an area surrounding the specified point.

METHODS

`public void` **componentHidden**`( ComponentEvent` **e** `)`

- **Usage**
  - ComponentListener interface method. Should not be called directly. Invoked when component has been hidden.
- **Parameters**
  - `e` - ComponentEvent

`public void` **componentMoved**`( ComponentEvent` **e** `)`

- **Usage**
  - ComponentListener interface method. Should not be called directly. Invoked when component has been moved.
- **Parameters**
  - `e` - ComponentEvent

`public void` **componentShown**`( ComponentEvent` **e** `)`

- **Usage**
  - ComponentListener interface method. Should not be called directly. Invoked when component has been shown.
- **Parameters**
  - `e` - ComponentEvent

`public MapContext` **getContext**`( )`

- **Returns** - the mapcontext

`public void` **layerChanged**`( LayerEvent` **e** `)`

`public abstract void` **paint**`( Graphics` **g** `)`

D.8 Package ie.tcd.cs.dsg.hermes.gis.generalisation.elimination

Package Contents *Page*

Classes

Class **EliminationFilter**

Instances of classes that implement this interface are used to filter Shapes. These instances are used to filter shapes based on the currently mimimum size shapes that are visible. i.e. if the pixel tolerance is set to 3 pixels then no shape that (when rendered) will occupy less than 3 x 3 pixels is accepted by the filter.
This is a very simple filtering of geometry based on its area and the current map scale. The filter keeps track of the current scale by listening to projection changes.

extends java.lang.Object
implements ie.tcd.cs.dsg.hermes.gis.geometry.ShapeListFilter

Fields

- public static final String LOD_ALGORITHM_PROPERTY
  - The properties file key used to determine which level of detail filter to load at runtime.
- public static final String LOD_TOLERANCE_PROPERTY
  - The key for the property that specifies the tolerance in pixels

- public static final String ELIMINATION_PROPERTY
  - Whether elimination should be used

Constructors

`public` **EliminationFilter**`( )`

Methods

`public static EliminationFilter` **getInstance**`( )`

- **Usage**
  - Gets an instance of a level of detail filter.
- **Returns** - an instance of a level of detail filter or NULL if an exception occurs

`public abstract void` **setScale**`( float` **f** `)`

- **Usage**
  - Sets the scale for filtering at. Must be called before a ShapeList is filtered.
- **Parameters**
  - `f` - the new scale

`public abstract void` **setTolerance**`( float` **minPixTolerance** `)`

- **Usage**
  - Sets the minimum number of pixels wide and high a shape in the map must be (at the current map scale) to be rendered.
- **Parameters**
  - `minPixTolerance` - the minimum number of pixels wide and high a shape in the map must be

CLASS **MinimumAreaFilter**

Filters Shape data based on its rendered area.

extends ie.tcd.cs.dsg.hermes.gis.generalisation.elimination.EliminationFilter

implements ie.tcd.cs.dsg.hermes.gis.tools.Algorithm

CONSTRUCTORS

`public` **MinimumAreaFilter**`( float` **pixTolerance** `)`

- **Usage**
  - Constructor. Filters Shape data based on its area.
- **Parameters**
  - `pixTolerance` - the minimum size of viewable shapes

METHODS

`public boolean` **accept**`( ShapeRecord` **record** `)`

- **Usage**
  - This method should never be called (Although it will work to some extent). Instead calls to accept() should specify the current scale as a parameter.
- **See Also**
  - `ShapeListFilter.accept(ShapeRecord)` (in D.6, page 271)

`public float` **getMinimumArea**`( )`

`public String` **getName**`( )`

`public void` **setScale**`( float` **f** `)`

`public void` **setTolerance**`( float` **minPixTolerance** `)`

D.9 Package ie.tcd.cs.dsg.hermes.gis.index.spatial

Package Contents *Page*

Interfaces

INTERFACE **SpatialIndex**

A generic interface to Spatial index's
implements ie.tcd.cs.dsg.hermes.gis.index.Index,
ie.tcd.cs.dsg.hermes.gis.tools.Algorithm

METHODS

`public void` **delete(** `IndexEntry` **entry** `)`

- **Usage**
  - Deletes the specified Entry from the index
- **Parameters**
  - `entry` - the index entry to delete

`public void` **delete(** `Rectangle` **mbr** `)`

- **Usage**
  - Deletes all entrys within the specified bounding box from the index.
- **Parameters**

  – `mbr` - The rectangle within which all entrys should be deleted

`public void` **`insert(`** `IndexEntry` **entry** `)`

- **Usage**
  – Inserts a single IndexEntry into the index.
- **Parameters**
  – `entry` - the entry to insert

`public void` **`insert(`** `ShapeList` **records** `)`

- **Usage**
  – Adds an index entry to this index for each of the shape records specified. Note. Assumes there are no duplicates in the list
- **Parameters**
  – `records` - a list of shape records

`public ShapeList` **`search(`** `Rectangle` **area** `)`

- **Usage**
  – Locates records in the shape file that intersect with the given rectangle. The spatial index is searched for intersections and the appropriate records are read from the shape file.
- **Returns** - a list of ShapeRecords with correct offsets and numbers

Classes

Class **RTree**

A disk based R-Tree Implementation. Uses only quadratic cost node splitting algorithm.

extends java.lang.Object

implements ie.tcd.cs.dsg.hermes.gis.tools.benchmark.IOAccounting

Fields

- public static final String NODE_SPLIT_PROPERTY
- public static final String SPLIT_QUADRATIC
  - Quadratic cost in number of entries per node
- public static final String SPLIT_LINEAR
  - Linear cost in number of entries per node

Constructors

`public` **RTree**`( PageStore` **store** `)`

- **Usage**
  - Constructor. R-Tree retrieved from store.
- **Parameters**
  - `store` - a disk-based data structure for storing the tree

Methods

`public void` **delete**`( Rectangle` **env** `)`

- **Usage**
  - Deletes the entry with the specified `Envelope` as its bounds. If more than one entry exists with the same bounds, then subsequent calls to `delete` are needed to remove all this elements.
- **Parameters**
  - `env` - The `Envelope`

`public Rectangle` **getBounds**`( )`

- **Usage**
  - Gets this index bounding box

- **Returns** - A Rectangle the bounding box of the root node

`public void` **`insert`**`( IndexEntry` **entry** `)`

- **Usage**
  - R-Tree insertion algorithm
- **Parameters**
  - `entry` - the IndexEntry to insert

`public ShapeList` **`search`**`( Rectangle` **query** `)`

- **Usage**
  - Performs a search on this `RTree`
- **Parameters**
  - `query` - the query `Envelope`
- **Returns** - a `Collection` of `Data`

Class **RTreeSpatialIndex**

A Spatial Index using a paged on disk R-Tree with Quadratic time Node splitting.

extends java.lang.Object
implements SpatialIndex

Fields

- public static final String PROPERTY_VALUE
  - The value of the SPATIAL_INDEX_PROPERTY if this class is to be used

Constructors

`public` **RTreeSpatialIndex**`( File` **pageStoreFile** `)`

- **Usage**
  - Constructor to create an index for reading from an already existing R-Tree.
- **Parameters**
  - `pageStoreFile` - a file in which an R-Tree is stored.

`public` **RTreeSpatialIndex**`( SHPFile` **shp**`, File` **pageStoreFile** `)`

- **Usage**
  - Constructor to create a new index for the specified shape file.
- **Parameters**
  - `shp` - the Shapefile containing geometry to be indexed

Methods

`public void` **delete**`( IndexEntry` **entry** `)`

`public void` **delete**`( Rectangle` **env** `)`

`public void` **insert**`( IndexEntry` **entry** `)`

`public void` **insert**`( ShapeList` **records** `)`

`public boolean` **isEmpty**`( )`

`public ShapeList` **search**`( Rectangle` **area** `)`

D.10 Package ie.tcd.cs.dsg.hermes.gis.layer

Package Contents *Page*

Interfaces

INTERFACE **MapLayer**

Layer objects are components which can be added to the MapContext to make a map.

Layers implement the ProjectionListener interface to listen for ProjectionEvents. When the projection changes, they may need to re-fetch, regenerate their graphics, and then repaint themselves into the MapCanvas.

implements ie.tcd.cs.dsg.hermes.gis.event.ProjectionListener, java.lang.Comparable

METHODS

`public void` **addLayerListener**`( LayerListener` **listener** `)`

- **Usage**
  - Adds a layer listener to the map layer. The map context should listen for changes in the layers it manages and pass events up to the canvas to trigger repaints.
- **Parameters**
  - `listener` - the listener

`public String` **getLayerName**`( )`

- **Usage**
  - Gets the layers descriptive name
- **Returns** - the layers name

`public Style` **getStyle**`( )`

- **Usage**
  - Gets the style currently being used by the layer to render vector graphics.
- **Returns** - the current rendering style

`public int` **getZOrder**`( )`

- **Usage**
  - Gets the layer rendering order
- **Returns** - the layer rendering order

`public boolean` **isDynamic**`( )`

- **Usage**
  - Tests to see if the current map layer is dynamic (i.e. contains rapidly changing data and is responsible for its own repainting) or static in which case double buffering will be used to speed up the rendering.

- **Returns** - true if the curren layer is an instance of DynamicMapLayer

`public boolean` **isVisible**`( )`

- **Usage**
  - Gets the layers visibility
- **Returns** - true is the layer is currently being drawn on the map canvas

`public void` **render**`( Graphics` **g** `)`

- **Usage**
  - Render the current layer to the specified graphics context.
- **Parameters**
  - `g` - a graphics context to render the layer to

`public void` **setMaxScale**`( float` **f** `)`

- **Usage**
  - Sets the maximum scale at which the currnet layer is visible
- **Parameters**
  - `f` - the scale in pixels per meter

`public void` **setMinScale**`( float` **f** `)`

- **Usage**
  - Sets the minimum scale at which the currnet layer is visible
- **Parameters**
  - `f` - the scale in pixels per meter

`public void` **setVisible**`( boolean` **visible** `)`

- **Usage**
  - Sets the layers visibility on the canvas
- **Parameters**

  – `visible` - boolean

`public void` **setZOrder**`( int` **zorder** `)`

- **Usage**
  - Sets the layer rendering order
- **Parameters**
  - `zorder` - the layer rendering order

Classes

Class **AbstractMapLayer**

All map layers must extend this class. This class implements the generic methods such as getLayerName() that are common to all map layers.

extends java.lang.Object
implements MapLayer

Constructors

`public` **AbstractMapLayer**`( String` **layerName**`, RotatableProjection` **projection**`, Style` **style**`, boolean` **dynamic** `)`

- **Parameters**
  - `layerName` -

Methods

`public void` **addLayerListener**`( LayerListener` **listener** `)`

- **Usage**
  - Adds a listener to this layer for changes in the contents of the layer.
- **Parameters**
  - `listener` - the listener to add

`public String` **getLayerName**`( )`

`public Style` **getStyle**`( )`

`public int` **getZOrder**`( )`

`public boolean` **isDynamic**`( )`

`public boolean` **isVisible**`( )`

`public void` **projectionChanged**`( ProjectionEvent` **e** `)`

`public void` **setMaxScale**`( float` **f** `)`

`public void` **setMinScale**`( float` **f** `)`

`public void` **setVisible**`( boolean` **visible** `)`

`public void` **setZOrder**`( int` **zorder** `)`

Class **AnnotationShapeMapLayer**

This layer draws text annotations on the map.

extends ie.tcd.cs.dsg.hermes.gis.layer.ShapeMapLayer

Constructors

`public` **AnnotationShapeMapLayer**`( String` **layerName**, `RotatableProjection` **projection**, `Style` **style**, `SpatialQuery` **spatialQuery** `)`

- **Usage**
  - Constructs a shape layer
- **Parameters**
  - `layerName` - the name of the layer
  - `spatialIndex` - the spatial index from which shapes can be retrieved
  - `spatialQuery` - the initial query to use to retrieve shapes from the index

Methods

`public void` **projectionChanged**`( ProjectionEvent` **evt** `)`

`public void` **render**`( Graphics` **g** `)`

Class **BufferedMapLayer**

Buffered map layers are drawn as a background and double buffered to prevent flickering.

extends ie.tcd.cs.dsg.hermes.gis.layer.AbstractMapLayer

Constructors

`public` **BufferedMapLayer**`( String` **layerName**`, RotatableProjection` **projection**`, Style` **style** `)`

- **Parameters**
  - `layerName` - the name of the layer
  - `projection` - the projection in use
  - `style` - the style definitions to use when rendering the layer

Class **CompassMapLayer**

Draws a north arrow (compass rose) on the screen. As the map is rotated the arrow will continue to point north by remaining in sync with the projection and updating in response to all projection changes.

extends ie.tcd.cs.dsg.hermes.gis.layer.DynamicMapLayer

Constructors

`public` **CompassMapLayer**`( String` **layerName**`, RotatableProjection` **projection**`, Style` **style** `)`

- **Parameters**
  - `layerName` -

METHODS

`public void` **projectionChanged**`( ProjectionEvent` **evt** `)`

`public void` **render**`( Graphics` **g** `)`

CLASS **DynamicMapLayer**

Abstract class representing MapLayers that display dynamic data and must always be redrawn in every rendering cycle. Buffering the contents of Dynamic Layers is not allowed as it would result in stale data being represented.

extends ie.tcd.cs.dsg.hermes.gis.layer.AbstractMapLayer

CONSTRUCTORS

`public` **DynamicMapLayer**`( String` **layerName**`, RotatableProjection` **projection**`, Style` **style** `)`

- **Parameters**
  - `layerName` - the name of the layer
  - `projection` - the projection in use
  - `style` - the style definitions to use when rendering the layer

CLASS **GraphMapLayer**

Renders a graph (road network) as a map layer. This layer was used to debug the route generation facilities in Mobile GIS. You would not generally want to overlay road centers on the map.

extends ie.tcd.cs.dsg.hermes.gis.layer.BufferedMapLayer

CONSTRUCTORS

`public` **GraphMapLayer**`( String` **layerName**`, RotatableProjection` **projection**`, Style` **style**`, Graph` **graph** `)`

- **Usage**
  - Constructs a shape layer
- **Parameters**
  - `layerName` - the name of the layer
  - `projection` - the projection for transforming between coordinate spaces
  - `style` - the drawing style for rendering
  - `graph` - the graph of nodes

Methods

`public Graph` **getGraph**`( )`

- **Returns** - Returns the graph.

`public void` **render**`( Graphics` **g** `)`

Class **GraticuleMapLayer**

Draws a square grid over the map similar to topological maps

extends ie.tcd.cs.dsg.hermes.gis.layer.BufferedMapLayer

Fields

- public static final int VERTICAL
- public static final int HORIZONTAL
- public static final int VERTICAL_AND_HORIZONTAL

Constructors

`public` **GraticuleMapLayer**`( String` **layerName**`, RotatableProjection` **projection**`, Style` **style**`, int` **threshold** `)`

- **Usage**
  - Constructs a Graticule shape layer
- **Parameters**
  - `layerName` - the name of the layer
  - `projection` - the projection for transforming between coodrinate spaces
  - `style` - the drawing style for rendering
  - `threshold` - the distance between graticule lines

METHODS

`public void` **projectionChanged**`( ProjectionEvent` **evt** `)`

`public void` **render**`( Graphics` **g** `)`

CLASS **MapLayer.ZOrder**

Z-Order Constants. MIDDLE specifies arbitrary ordering and should be the default. When more than one MapLayer is told to be on TOP the result is not guaranteed.

extends java.lang.Object

FIELDS

- public static final int TOP
- public static final int BOTTOM
- public static final int MIDDLE

CONSTRUCTORS

`public` **MapLayer.ZOrder**`( )`

Class **ScaleMapLayer**

Shows the map scale in the bottom left of the map panel

extends ie.tcd.cs.dsg.hermes.gis.layer.BufferedMapLayer

Constructors

`public` **ScaleMapLayer**`( String` **layerName**`, RotatableProjection` **projection**`, Style` **style** `)`

- **Usage**
  - Constructs a new Scale map layer using the specified projection to retrieve scale information from and position the text on screen. The text is rendered according to the rules in the specified Style.
- **Parameters**
  - `layerName` - the name of the layer
  - `projection` - the projection
  - `style` - the rendering style

Methods

`public void` **projectionChanged**`( ProjectionEvent` **evt** `)`

`public void` **render**`( Graphics` **g** `)`

Class **ShapeMapLayer**

A layer containing ESRI Shapefile data retrieved using a spatial query on a specified query engine.

extends ie.tcd.cs.dsg.hermes.gis.layer.BufferedMapLayer

Constructors

`public` **ShapeMapLayer**`(` `String` **layerName**, `RotatableProjection` **projection**, `Style` **style**, `SpatialQuery` **spatialQuery** `)`

- **Usage**
  - Constructs a shape layer
- **Parameters**
  - `layerName` - the name of the layer
  - `spatialIndex` - the spatial index from which shapes can be retrieved
  - `spatialQuery` - the initial query to use to retrieve shapes from the index

Methods

`public SpatialQuery` **getSpatialQuery**`( )`

- **Usage**
  - The query specifying the data that is represented on this layer
- **Returns** - The query used to select which records from the spatial index are chosen to be displayed in this layer

`public void` **projectionChanged**`(` `ProjectionEvent` **evt** `)`

`public void` **render**`(` `Graphics` **g** `)`

Class **VisibilityShapeMapLayer**

This layer illustrates which objects are visible by colouring them red. This layer was developed to illustrate the operation of the Visibility algorithms.

extends ie.tcd.cs.dsg.hermes.gis.layer.DynamicMapLayer
implements ie.tcd.cs.dsg.hermes.context.location.LocationListener

Constructors

`public` **VisibilityShapeMapLayer**`(` `String` **layerName**, `RotatableProjection` **projection**, `Style` **style**, `SpatialQuery` **spatialQuery** `)`

- **Usage**
  - Constructs a shape layer
- **Parameters**
  - `layerName` - the name of the layer
  - `spatialIndex` - the spatial index from which shapes can be retrieved
  - `spatialQuery` - the initial query to use to retrieve shapes from the index

Methods

`public void` **locationUpdated**`( Location` **location** `)`

`public void` **projectionChanged**`( ProjectionEvent` **evt** `)`

`public void` **render**`( Graphics` **g** `)`

D.11 Package ie.tcd.cs.dsg.hermes.gis.projection.model

Package Contents *Page*

Classes

Class **WorldModel**

Contains the constants the define how the world is modelled.

extends java.lang.Object

Fields

- public static WorldModel EARTH
  - The planet Earth
- public static WorldModel MARS
  - The planet Mars
- public static WorldModel TESTMODEL
  - A test model for checking geometry. Operates as if flat
- public final String key

Methods

`public static float` **getDateline**`( )`

- **Returns** - the dateline position

`public static final double` **getDistanceToHorizon**`( int` **heightAboveSurface** `)`

- **Usage**
  - The straight, line-of-sight distance (from your eyes to the horizon). For small heights above sea level and certainly for heights within the Earth's atmosphere the difference between this and the actual distance along the planet's surface to the horizon is small.

    See:

    http://newton.ex.ac.uk/research/qsystems/people/sque/physics/horizon/

    The straight-line distance to the horizon equation: ds = sqrt(h(2r + h)) where r = planet radius
- **Parameters**
  - `heightAboveSurface` - height above surface of planet in meters
- **Returns** - the distance in km

`public static float` **getLonRange**`( )`

- **Returns** - the max longitude

`public static float` **getNorthPole**`( )`

- **Returns** - the north pole latitude

`public static final float` **getPlanetEquatorialCircumference**`( )`

- **Returns** - the planets equatorial circumference in meters

`public static final float` **getPlanetRadius**`( )`

- **Returns** - the planet radius (in meters?)

`public static float` **getSouthPole**`( )`

- **Returns** - the south pole latitude

`public static final float` **radiansToMeters**`( float` **distRad** `)`

- **Usage**
  - Used to convert between a spherical distance in radians to meters.
- **Parameters**
  - `distRad` - distance in radians (see GreatCircle.spherical_distance)
- **Returns** - the arc radians converted to meters

D.12 Package ie.tcd.cs.dsg.hermes.gis.graph

Package Contents *Page*

Interfaces

Interface **Edge**

Represents an edge in Graph. An edge is an arc in a graph which connects exactly two nodes. These two nodes are referred to as the A node and the B node of the edge. The order of the A node and the B node is referred to as the **node orientation** of the edge.

implements Graphable

Methods

`public Node` **getNodeA( )**

- **Usage**
  - Returns the A node of the edge.
- **Returns** - The A node.

`public Node` **getNodeB( )**

- **Usage**
  - Returns the B node of the edge.
- **Returns** - The B node.

`public Node` **getOtherNode(** `Node` **node)**

- **Usage**
  - Returns one of the two nodes of an edge. If the specified node is node A, then node B is returned, and vice versa.
- **Parameters**
  - `node` - The node opposite of the node to return.
- **Returns** - Node A if node B is specified, node B if node A is specified.

INTERFACE **Graph**

Represents a graph, which is a collection of nodes (verticies) connected by links called edges (arcs).

The Graph object is intended to serve as a container for a collection of nodes and edges. It does don't define or manage the relationship among the components it contains.

METHODS

`public boolean` **contains(** `Node` **n)**

- **Usage**
  - Tests to see if the graph contains the specified node.
- **Parameters**
  - `n` - the node to test
- **Returns** - true if the specified node is in this graph

`public Edge` **getEdges( )**

- **Usage**
  - Returns the edges of the graph.
- **Returns** - A collection of Edge objects.
- **See Also**
  - `ie.tcd.cs.dsg.hermes.gis.graph.Edge` (in D.12, page 317)

`public Node` **getNodes**`( )`

- **Usage**
  - Returns the nodes of the graph.
- **Returns** - A collection of Node objects.
- **See Also**
  - `ie.tcd.cs.dsg.hermes.gis.graph.Node` (in D.12, page 320)

`public List` **getNodesOfDegree**`( int` **n** `)`

- **Usage**
  - Returns all the nodes in the graph of a specified degree. The degree of a node is the number of edges that are adjacent to the node.
- **Parameters**
  - `n` - The desired degree of nodes to be returned.
- **Returns** - A collection of nodes of degree n.
- **See Also**
  - `ie.tcd.cs.dsg.hermes.gis.graph.Node.getDegree()`

`public List` **getVisitedEdges**`( boolean` **visited** `)`

- **Usage**
  - Returns all the edges in the graph that have been marked as visited or non-visited.
- **Parameters**
  - `visited` - True if edge is visited, false if edge is unvisited.

- **Returns** - List of edges marked as visited / non-visited.
- **See Also**
  - `ie.tcd.cs.dsg.hermes.gis.graph.Graphable.isVisited()`

`public List` **getVisitedNodes**`( boolean` **visited** `)`

- **Usage**
  - Returns all the nodes in the graph that have been marked as visited or non-visited.
- **Parameters**
  - `visited` - True if node is visited, false if node is unvisited.
- **Returns** - List of nodes marked as visited / non-visited.
- **See Also**
  - `ie.tcd.cs.dsg.hermes.gis.graph.Graphable.isVisited()`

Interface **Node**

Represents a node in a graph. A node is a point in a graph which is adjacent to 0 or more edges. The collection of edges that are incident/ adjacent to the node, is referred to as the "adjacency list" of the node.

implements Graphable

Methods

`public void` **add**`( Edge` **e** `)`

- **Usage**
  - Adds an edge to the adjacency list of the node.
- **Parameters**
  - `e` - Adjacent edge to add.

`public boolean` **connectsWithEdge**`( Edge` **e** `)`

- **Usage**
  - Tests to see if the specified edge is already connected with this node.
- **Parameters**
  - `e` - the edge to test
- **Returns** - true if this edge is connected to this node

`public int` **getDegree**`( )`

- **Usage**
  - Returns the degree of the node. The degree of a node is defined as the number of edges that are adjacent to the node.
- **Returns** - int Degree of node.

`public Edge` **getEdge**`( Node` **other** `)`

- **Usage**
  - Returns an edge in the adjacency list of the node that is adjacent to another specified node. Note: It is possible for two nodes to share multiple edges between them. In this case, getEdges(Node other) can be used to obtain a complete list.
- **Parameters**
  - `other` - The other node that the desired edge to return is adjacent to.
- **Returns** - The first edge that is found to be adjacent to the specified node.

`public Edge` **getEdges**`( )`

- **Usage**
  - Returns the edge adjacency list of the node.
- **Returns** - A list containing all edges that are adjacent to the node.

`public List` **getEdges**`( Node` **other** `)`

- **Usage**

  – Returns a collection of edges in the adjacency list of the node that are adjacent to another specified node.

- **Parameters**
  – `other` - The other node that the desired edges to return are adjacent to.
- **Returns** - List of all edges that are found to be adjacent to the specified node.

`public void` **remove**`( Edge` **e** `)`

- **Usage**
  – Removes an edge from the adjacency list of the node.
- **Parameters**
  – `e` - Adjacent edge to remove.

Classes

Class BasicEdge

Represents a graph edge consisting of two nodes.

extends ie.tcd.cs.dsg.hermes.gis.geometry.Polyline
implements Edge

Constructors

`public` **BasicEdge**`( Polyline` **poly** `)`

- **Parameters**
  – `poly` - a line from which the start and end points will be used

Methods

`public boolean` **equals**`( Object` **o** `)`

- **Usage**

  - Points are equals when they have the same coordinates.

    Note: Z coordinates are not compared!

- **See Also**
  - `java.lang.Object.equals(java.lang.Object)`

`public Node` **getNodeA**`( )`

`public Node` **getNodeB**`( )`

`public Node` **getOtherNode**`( Node` **node** `)`

`public boolean` **isVisited**`( )`

`public void` **render**`( Graphics` **g**, `RotatableProjection` **projection**, `Style` **style** `)`

`public void` **setNodeA**`( Node` **a** `)`

- **Usage**
  - Overwrites the reference to the start node of this edge with the specified node. Required to perform clustering of nodes and merging of edges.
- **Parameters**
  - `a` - A node to replace the current node with

`public void` **setNodeB**`( Node` **b** `)`

- **Usage**
  - Overwrites the reference to the end node of this edge with the specified node. Required to perform clustering of nodes and merging of edges.
- **Parameters**
  - `b` - A node to replace the current node with

`public void` **setVisited**`( boolean` **visited** `)`

Class **BasicGraph**

An implementation of the Graph interface.

extends java.lang.Object

implements Graph

CONSTRUCTORS

`public` **BasicGraph(** `Node []` **nodes**, `Edge []` **edges**)

- **Parameters**
  - `nodes` - The nodes of the graph (clustered)
  - `edges` - The edges of the graph (redundant edges removed)

METHODS

`public boolean` **contains(** `Node` **n**)

`public Edge` **getEdges(**)

`public Node` **getNodes(**)

`public List` **getNodesOfDegree(** `int` **n**)

`public List` **getVisitedEdges(** `boolean` **visited**)

`public List` **getVisitedNodes(** `boolean` **visited**)

CLASS BasicNode

Represents a node in a graph

extends ie.tcd.cs.dsg.hermes.gis.geometry.Point

implements Node

CONSTRUCTORS

`public` **BasicNode(** `float` **x**, `float` **y**)

- **Usage**
  - Constructs a node with the specified coordinates
- **Parameters**

- `x` - the coordinate at which the node is located
- `y` - the coordinate at which the node is located

Methods

`public void` **add**`(` `Edge` **e** `)`

`public boolean` **connectsWithEdge**`(` `Edge` **e** `)`

`public int` **getDegree**`( )`

`public Edge` **getEdge**`(` `Node` **other** `)`

`public Edge` **getEdges**`( )`

`public List` **getEdges**`(` `Node` **other** `)`

`public boolean` **isVisited**`( )`

`public void` **remove**`(` `Edge` **e** `)`

`public void` **render**`(` `Graphics` **g**, `RotatableProjection` **projection**, `Style` **style** `)`

`public void` **setVisited**`(` `boolean` **visited** `)`

Class **TopologicalModelBuilder**

This class discovers topoligical relationships from line intersections in the underlying geometry of the map. Providing a list of polylines (representing roads) this class will cluster nodes and build a topological graph of the road network.

extends java.lang.Object

Constructors

`public` **TopologicalModelBuilder**`( )`

Methods

`public double` **getClusteringTolerance**`( )`

- **Returns** - Returns the clusteringTolerance.

`public Graph` **getTopologyGraph**`( ShapeList` **shapeList** `)`

`public void` **setClusteringTolerance**`( double` **clusteringTolerance** `)`

- **Parameters**
  - `clusteringTolerance` - The clusteringTolerance to set.

D.13 Package ie.tcd.cs.dsg.hermes.gis.graph.traverse

Package Contents *Page*

Interfaces

Interface **EdgeWeighter**

Supplies a weight for each edge in the graph to be used by the iteration when calculating node costs.

Methods

`public double` **getWeight**`( Edge e )`

- **Usage**
  - Returns the weight for the associated edge.
- **Parameters**
  - `e` - The edge whose weight to return.
- **Returns** - The weight of the edge.

Classes

Class **DijkstraNode**

Extends a graph node to allow the labelling of a node with it's shortest distance from the source node.

extends ie.tcd.cs.dsg.hermes.gis.graph.BasicNode

Constructors

`public` **DijkstraNode**`( float` **lat**`, float` **lon** `)`

Methods

`public double` **getLabel**`( )`

- **Returns** - Returns the label.

`public String` **getNodeName**`( )`

- **Returns** - Returns the nodeName.

`public void` **render**`( Graphics` **g**`, RotatableProjection` **projection**`, Style` **style** `)`

`public void` **setLabel**`( double` **label** `)`

- **Parameters**
  - `label` - The label to set.

`public void` **setNodeName**`( String` **nodeName** `)`

- **Parameters**
  - `nodeName` - The nodeName to set.

Class **DijkstraShortestPathFinder**

Calculates node paths in a graph using Dijkstra's Shortest Path Algorithm.

extends java.lang.Object

Constructors

`public` **DijkstraShortestPathFinder**`(` `Graph` **graph**`,` `EdgeWeighter` **edgeWeighter** `)`

- **Parameters**
  - `graph` - the graph to search
  - `edgeWeighter` - an edge weighing algorithm

Methods

`public Path` **getPath**`(` `Node` **source**`,` `Node` **target** `)`

Class **Path**

Represents a walk in a graph. A **walk** W is defined as an ordered set of nodes that two adjacent nodes in the set share an edge.

extends java.util.LinkedList

Constructors

`public` **Path**`( )`

Methods

`public boolean` **add**`(` `Node` **n** `)`

- **Usage**
  - Adding a new node clears the current list of edges.

- **Parameters**
  - `n` -

`public List` **getEdges**`( )`

- **Returns** - a list of the edges on the path

`public boolean` **isValid**`( )`

- **Usage**
  - Tests if the path is valid. A valid path satisfies two conditions: 1.) Each pair of adjacent nodes share an edge; 2.) There are no node repetitions.

Class **ShortestPathEdgeWeighter**

Gets the weight for an edge in a graph by calculating the distance between the nodes at either end of the edge.

extends java.lang.Object
implements EdgeWeighter

Constructors

`public` **ShortestPathEdgeWeighter**`( )`

Methods

`public double` **getWeight**`( Edge  e )`

- **Usage**
  - This implementation does not calculate the real length of the edge but instead makes an approximation by getting the distance between the two end points of the edge.

    Note: Edge weights must be positive for Dijkstra's Algorithm

D.14 Package ie.tcd.cs.dsg.hermes.gis.projection

Package Contents *Page*

Interfaces

INTERFACE **Projection**

A projection is an object that is maintained by the map, and represents a abstract "view" of the data. The projection has the properties of x-width, * y-height, scale (in pixels/meters), and latitude/longitude center point. At the center point of the projection, North is to the top of the screen.

implements java.lang.Cloneable, ie.tcd.cs.dsg.hermes.gis.tools.Algorithm

FIELDS

- public static final String PROJECTION_MODEL_PROPERTY
- public static final String PROJECTION_PROPERTY

METHODS

`public Point` **getCenter**`( )`

- **Usage**
  - Get the center LatLonPoint.
- **Returns** - center point

`public int` **getHeight**`( )`

- **Usage**
  - Get the height of the map.
- **Returns** - int height.

`public Point` **getLowerRight**`( )`

- **Usage**
  - Get the lower right (southeast) point of the projection.

    Returns the lower right point (or closest equivalent) of the projection based on the center point and height and width of screen.

    This is trivial for most cylindrical projections, but much more complicated for azimuthal projections.
- **Returns** - LatLonPoint

`public float` **getPixelsPerDegree**`( )`

- **Usage**
  - Used to calculate line simplification and generalisation tolerances as well as internally within the projection for scaling.
- **Returns** - the number of pixels in a degree

`public float` **getScale**`( )`

- **Usage**
  - Get the scale.

- **Returns** - float scale

`public float` **getScale**`( Point` **ll1**`, Point` **ll2**`, Screen.Pixel` **point1**`, Screen.Pixel` **point2** `)`

- **Usage**
  - Given a couple of points representing a bounding box, find out what the scale should be in order to make those points appear at the corners of the projection.
- **Parameters**
  - `ll1` - the upper left coordinates of the bounding box.
  - `ll2` - the lower right coordinates of the bounding box.
  - `point1` - a java.awt.Point reflecting a pixel spot on the projection that matches the ll1 coordinate, the upper left corner of the area of interest.
  - `point2` - a java.awt.Point reflecting a pixel spot on the projection that matches the ll2 coordinate, usually the lower right corner of the area of interest.

`public Point` **getUpperLeft**`( )`

- **Usage**
  - Get the upper left (northwest) point of the projection.

    Returns the upper left point (or closest equivalent) of the projection based on the center point and height and width of screen.

    This is trivial for most cylindrical projections, but much more complicated for azimuthal projections.
- **Returns** - LatLonPoint

`public int` **getWidth**`( )`

- **Usage**
  - Get the width of the map.

- **Returns** - int width.

`public Rectangle` **getWorldExtent**`( )`

- **Usage**
  - Get the bounding box of the projection.
- **Returns** - BBox bounds in lat / lon degrees for map

`public WorldModel` **getWorldModel**`( )`

- **Returns** - the world model in use

`public Point` **pixelToWorld**`( int` **x**`, int` **y** `)`

`public void` **setCenter**`( Point` **center** `)`

- **Usage**
  - Set center point of projection.

`public void` **setScale**`( float` **s** `)`

- **Usage**
  - Sets the projection to the scale 1:s if minscale <s <maxscale.
- **Parameters**
  - `s` - float scale

`public void` **setScale**`( Rectangle` **box** `)`

- **Usage**
  - Given a bounding box, set the scale of the projection to what it should be in order to make those points appear at the corners of the projection.
- **Parameters**
  - `box` - the new world extents

`public void` **setScreenSize**`( int` **width**`, int` **height** `)`

- **Usage**

  – Sets the screen width

- **Parameters**
  - `width` -

`public Screen.Pixel` **worldToPixel**`( float` **lat**`, float` **lon** `)`

- **Usage**
  - Forward project lat,lon coordinates (in degrees) into xy space.
- **Parameters**
  - `lat` - float latitude in decimal degrees
  - `lon` - float longitude in decimal degrees decimal degrees
- **Returns** - Point (new)

Interface **RotatableProjection**

Rendering engines must implement this interface. A coordinate space transformation that supports rotating of the screen transformation space after projecting.

implements Projection

Methods

`public boolean` **equalsIgnoreRotation**`( RotatableProjection` **proj** `)`

- **Usage**
  - Tests to see if the specified rotation is equivalent to this projection is the rotation degrees are ignored. Used to test if a cached projection value is still valid.
- **Parameters**
  - `proj` - the current projection
- **Returns** - true if the projection are the same except for rotation

`public float` **getRotationDegrees**`( )`

- **Usage**
  - Returns the degrees the map is rotated by from north
- **Returns** - the degrees the map is rotated by from north

`public Point` **pixelToWorld**`( int` **x**`, int` **y**`, boolean` **rotate** `)`

- **Usage**
  - Inverse project x,y coordinates. The boolean parameter control whether the rotation should also be applied. This is used when projecting bounding box coordinates which should never be rotated and must always remain rectangular.
- **Parameters**
  - `x` - pixel x component
  - `y` - pixel y component
  - `rotate` - rotate the coordinates before projecting
- **Returns** - LatLonPoint (new)

`public void` **rotateByDegrees**`( double` **rotationDegrees** `)`

- **Usage**
  - Appends the specified number of degrees rotation to the maps current rotation angle.
- **Parameters**
  - `rotationDegrees` - The rotationDegrees to add.

`public void` **rotateToDegrees**`( double` **rotationDegrees** `)`

- **Usage**
  - Sets the angle in degrees that the map should be rotated (Clockwise) by.
- **Parameters**
  - `rotationDegrees` - The rotationDegrees to set.

`public Screen.Pixel` **rotateToPixel**`( Screen.Pixel` **pixel** `)`

- **Usage**
  - Rotates the specified pixel about he screen center and returns the new point. No projection transformation is performed.
- **Parameters**
  - `pixel` - the pixel to rotate to the screens coordinate space
- **Returns** - the rotated pixcel

`public Screen.Pixel` **rotateToWorld**`( Screen.Pixel` **pixel** `)`

- **Usage**
  - Rotates the specified pixel in reverse about he screen center and returns the new point. No projection transformation is performed. This results in the North orientated point on the map before rotation was performed.
- **Parameters**
  - `pixel` - the pixel to rotate to the world coordinate space
- **Returns** - the rotated pixcel

`public Screen.Pixel` **worldToPixel**`( float` **lat**`, float` **lon**`, boolean` **rotate** `)`

- **Usage**
  - Forward project lat,lon coordinates (in degrees) into xy space. The boolean parameter controls whether the rotation should also be applied. This allows rotation to be separated form projection allowing projections to be cached, thus speeding up the rotating of the map.
- **Parameters**
  - `lat` - float latitude in decimal degrees
  - `lon` - float longitude in decimal degrees decimal degrees
  - `rotate` - rotate the coordinates after projecting
- **Returns** - Point (new)

Classes

Class **AbstractProjection**

AbstractProjection is the base class of all Projections. You probably don't want to use this class unless you are hacking your own projections, or need extended functionality.
The worldToPixel() and pixelToWorld() methods are currently implemented using the algorithms given in John Synder's *Map Projections –A Working Manual* for the sphere. This is sufficient for display purposes.

extends java.lang.Object
implements RotatableProjection

Constructors

`public` **AbstractProjection**`( Point` **center**`, float` **s**`, int` **w**`, int` **h** `)`

- **Usage**
  – Construct a projection.
- **Parameters**
  – `center` - LatLonPoint center of projection
  – `s` - float scale of projection
  – `w` - width of screen
  – `h` - height of screen

Methods

`public final boolean` **equalsIgnoreRotation**`( RotatableProjection` **proj** `)`

`public final Point` **getCenter**`( )`

`public final int` **getHeight**`( )`

`public final Point` **getLowerRight**`( )`

`public final float` **getPixelsPerDegree**`( )`

`public final float` **getRotationDegrees**`( )`

`public final float` **getScale**( )

`public float` **getScale**(`Point` **ll1**, `Point` **ll2**, `Screen.Pixel` **point1**, `Screen.Pixel` **point2**)

`public final Point` **getUpperLeft**( )

`public final int` **getWidth**( )

`public final Rectangle` **getWorldExtent**( )

- **Usage**
  - Gets the world extents that is visible. Note that a slightly larger world extent is returned to make sure enough geometry is retrieved to draw a rotated map without clipped shapes becoming visible.
- **See Also**
  - `ie.tcd.cs.dsg.hermes..gis..projection.Projection.getWorldExtent()`

`public WorldModel` **getWorldModel**( )

- **Returns** - the world model in use

`public abstract Point` **pixelToWorld**(`int` **x**, `int` **y**)

`public Point` **pixelToWorld**(`Screen.Pixel` **pc**)

`public void` **rotateByDegrees**(`double` **rotationDegrees**)

`public void` **rotateToDegrees**(`double` **rotationDegrees**)

`public final Screen.Pixel` **rotateToPixel**(`Screen.Pixel` **pixel**)

`public final Screen.Pixel` **rotateToWorld**(`Screen.Pixel` **pixel**)

`public final void` **setCenter**(`Point` **center**)

`public final void` **setScale**(`float` **s**)

`public final void` **setScale**(`Rectangle` **box**)

`public final void` **setScreenSize**(`int` **width**, `int` **height**)

`public abstract Screen.Pixel` **worldToPixel**(`float` **lat**, `float` **lon**)

Class **EquiRectangular**

Implements the EquiRectangularProjection projection, which is basically something where the lat/lon and pixel ratios are the same.
EquiRectangular projections arealso known as Equidistant Cyllindrical.

extends ie.tcd.cs.dsg.hermes.gis.projection.AbstractProjection

Constructors

`public` **EquiRectangular**`( Point` **center**`, float` **s**`, int` **w**`, int` **h** `)`

- **Usage**
  - Construct a projection.
- **Parameters**
  - `center` - LatLonPoint center of projection
  - `s` - float scale of projection
  - `w` - width of screen
  - `h` - height of screen

Methods

`public final Point` **pixelToWorld**`( int` **x**`, int` **y** `)`

`public final Point` **pixelToWorld**`( int` **x**`, int` **y**`, boolean` **rotate** `)`

`public final Screen.Pixel` **worldToPixel**`( float` **lat**`, float` **lon** `)`

`public final Screen.Pixel` **worldToPixel**`( float` **lat**`, float` **lon**`, boolean` **rotate** `)`

Class **Mercator**

Implements the Mercator projection. A conformal projection, angles are preserved around all locations, however scale varies from place to place, distorting the size of geographical objects. In particular, areas closer to the poles are more affected, transmitting an image of the geometry of the planet which is more distorted the closer to the poles.

extends ie.tcd.cs.dsg.hermes.gis.projection.AbstractProjection

Constructors

`public` **Mercator**`( Point` **center**`, float` **scale**`, int` **width**`, int` **height** `)`

- **Usage**
  - Construct a Mercator projection.
- **Parameters**
  - `center` - LatLonPoint center of projection
  - `scale` - float scale of projection
  - `width` - width of screen
  - `height` - height of screen

Methods

`public Point` **pixelToWorld**`( int` **x**`, int` **y** `)`

`public Point` **pixelToWorld**`( int` **xx**`, int` **yy**`, boolean` **rotate** `)`

`public Screen.Pixel` **worldToPixel**`( float` **lat**`, float` **lon** `)`

`public Screen.Pixel` **worldToPixel**`( float` **lat**`, float` **lon**`, boolean` **rotate** `)`

Class **Screen**

Models Screen dimensions

extends java.lang.Object

Fields

- public int _xmax
  - Maximum X value - Bottom right x coordinate

- public int _xmin
  - Minimum X value - Top left x coordinate
- public int _ymax
  - Maximum Y value - Bottom right y coordinate
- public int _ymin
  - Minimum Y value - Top left y coordinate

Constructors

`public` **Screen**`( int` **xmin**`, int` **ymin**`, int` **xmax**`, int` **ymax** `)`

- **Usage**
  - constructor

Methods

`public Screen.Pixel` **getLowerRight**`( )`

- **Usage**
  - The lower right point
- **Returns** - the lower right point

`public static final Screen` **getSafeToRotate**`( Screen s )`

- **Usage**
  - Calculates a screen size where the width and height are at least as long as the diagonal of this screen. Used to generate a clipping area that is large enough to survive rotating without displaying the clipped edges on screen.
- **Returns** - a new Screen dimension

`public Screen.Pixel` **getUpperLeft**`( )`

- **Usage**

  – The upper left point

- **Returns** - The upper left point

`public void` **grow**`( int` **per** `)`

- **Usage**
  – Increase the size of the bounding box by adding a the specified amounts to the existing box on all sides.
- **Parameters**
  – `per` - the percentage to shrink the box by

`public void` **shrink**`( int` **per** `)`

- **Usage**
  – Decrease the size of the bounding box by adding a the specified amounts to the existing box on all sides.
- **Parameters**
  – `per` - the percentage to shrink the box by

D.15 Package ie.tcd.cs.dsg.hermes.gis.query

Package Contents *Page*

Interfaces

Interface Constraint

A Query Constraint must implement this interface

Methods

`public int` **getOperator**`( )`

- **Usage**
  - Gets the operator
- **Returns** - one of the Operator constants

`public void` **setOperator**`( int` **operator** `)`

- **Usage**
  - Sets the operator
- **Parameters**
  - `operator` - one of the Operator constants

Interface **Operator**

Query Constraint Operators

Fields

- public static final int EQ
  - Equal to
- public static final int NOT
  - Not Equal
- public static final int LIKE
  - Equal (ignore case and white space)
- public static final int GREATER
  - Greater than
- public static final int LESS
  - Less than
- public static final int CONTAINS
  - Contained by area
- public static final int INTERSECTS
  - Intersects area

INTERFACE **Query**

All queries implement this interface.

FIELDS

- public static final int MAX_ATTRIBUTES
  - The maximum number of values in a constraint and maximum number of constraints in a query

METHODS

`public void` **addConstraint**`( Constraint` **c** `)`

- **Usage**
  - Adds the specified constraint to the query
- **Parameters**
  - `c` - the constraint to add

`public boolean` **constainsConstraint**`( Class` **constraintType** `)`

- **Usage**
  - Tests to see if a constraint of the specified type has been added to the query.
- **Parameters**
  - `constraintType` -
- **Returns** - true if a constraint of the type specified is part of the query

`public Constraint` **getConstraint**`( Class` **constraintType** `)`

- **Usage**
  - Gets all the constraints of type specified by class name.

- **Parameters**
  - `constraintType` - a constraint class name
- **Returns** - an array of constrains of type specified in query

`public boolean` **`validate( )`**

- **Usage**
  - Validates that a complete query syntactically correct query has been constructed. This will usually involve ensuring that required constraints have been added.
- **Returns** - true if the query is syntactically correct

INTERFACE **QueryEngine**

An interface defining the public access to query engine implementations. QueryEngines are responsible for managing the currently available data sources and executing queries on those data sources.

METHODS

`public void` **`addShapeDataSource( String type, String source )`**

- **Usage**
  - Adds the specified data source to the list of sources that all spatial queries will be executed on.
- **Parameters**
  - `type` - the type of data source to read. Options currently are 'SHP' or 'TEST' (lowercase!)
  - `source` - a Shape data source

`public void` **`clearCache( )`**

- **Usage**

  - Clears all cached records from data sources record cache. Used by benchmarking code to ensure fresh geometry is returned from queries.

    Note: Will raise an error is called on multiscale data as caches are maintained for each scale and must be cleared individually.

`public ShapeList` **executeQuery**`( SpatialQuery` **query** `)`

- **Usage**
  - Executes the specified query on all the data sources that have been added.

    Calls `executeQuery(SpatialQuery, SpatialDataSource)` in a loop
- **Parameters**
  - `query` - the spatial query to execute
- **Returns** - a list of all the shapes matching the query

`public ShapeList` **executeQuery**`( SpatialQuery` **query**`, SpatialDataSource` **source** `)`

- **Usage**
  - Executes the specified spatial query on the specified data source
- **Parameters**
  - `query` - the spatial query to execute
  - `source` - the data source to query
- **Returns** - a list of all the shape records (with geometry) that match the query

`public SpatialDataSource` **getShapeDataSource**`( String` **name** `)`

- **Usage**
  - Retrieves a data source based on it's name (As determined by `SpatialDataSource.getName()`)
- **Parameters**
  - `name` - the data sources name
- **Returns** - the data source or null if it was not found

`public Rectangle` **getWorldExtent**`( )`

- **Usage**
  - Gets the extent of all the data sources combined.
- **Returns** - the extents of all the current data sources combined

`public void` **removeAllShapeDataSources**`( )`

- **Usage**
  - Empties all data sources from the QueryEngine.

`public void` **removeShapeDataSource**`( SpatialDataSource` **source** `)`

- **Usage**
  - Removes the specified data source and updates the extents querable to reflect this change in data sources. Removing the data source from the list causes the `close()` method to be calls on the data source.
- **Parameters**
  - `source` - the data source to remove (must be same instance)

Classes

CLASS **AttributeConstraint**

Query attribute constraints constrain a particular fields allowable values.

extends java.lang.Object
implements Constraint, Operator

CONSTRUCTORS

`public` **AttributeConstraint**`( String` **field**`, Object` **value**`, int` **operator** `)`

- **Parameters**

- `field` - the field that is being constrained
- `value` - the value
- `operator` - the operator defined an allowed relationship between the field and specified value

METHODS

`public void` **addValue**`( Object` **value** `)`

- **Usage**
  - Adds a new possible value for the constrained field.
- **Parameters**
  - `value` -

`public String` **getField**`( )`

- **Usage**
  - Gets the field (the name of the attribute we are constraining)
- **Returns** - the name of the attribute

`public int` **getOperator**`( )`

- **Usage**
  - Gets the operator. Should be one of the static constants in the `Operator` class.
- **Returns** - the operator

`public Object` **getValues**`( )`

- **Usage**
  - Trims and returns the values array as an array of objects. Objects will be either Strings or Doubles
- **Returns** - all the values that are constraints of the field

`public void` **setOperator**`( int` **operator** `)`

- **Usage**
  - Sets the Constraint operator
- **Parameters**
  - `operator` -

Class **MobileGISQueryEngine**

This class manages the open data sources and processes spatial queries.

extends java.lang.Object

implements QueryEngine

Fields

- public EliminationFilter lod
  - Level of Detail scale filter

Methods

`public void` **addShapeDataSource**`( String` **type**`, String` **source** `)`

- **Usage**
  - Adds the specified data source to the list of sources that all spatial queries will be executed on.
- **Parameters**
  - `source` - a Shape data source

`public void` **clearCache**`( )`

`public ShapeList` **executeQuery**`( SpatialQuery` **query** `)`

- **Usage**
  - Executes the specified query on all the data sources that have been added. Calls `executeQuery(SpatialQuery, SpatialDataSource)` in a loop

- **Parameters**
  - `query` - the spatial query to execute
- **Returns** - a list of all the shapes matching the query

`public ShapeList` **executeQuery**`( SpatialQuery` **query**`, SpatialDataSource` **source** `)`

- **Usage**
  - Executes the specified spatial query on the specified data source
- **Parameters**
  - `query` - the spatial query to execute
  - `source` - the data source to query
- **Returns** - a list of all the shape records (with geometry) that match the query

`public SpatialDataSource` **getShapeDataSource**`( String` **name** `)`

- **Usage**
  - Retrieves a data source based on it's name (As determined by `SpatialDataSource.getName()`)
- **Parameters**
  - `name` - the data sources name
- **Returns** - the data source or null if it was not found

`public Rectangle` **getWorldExtent**`( )`

`public void` **removeAllShapeDataSources**`( )`

- **Usage**
  - Emptys all data sources from the QueryEngine. (And closes data sources)

`public void` **removeShapeDataSource**`( SpatialDataSource` **source** `)`

- **Usage**
  - Removes the specified data source and updates the extents querable to reflect this change in data sources. Removing the data source from the list causes the `close()` method to be calles on the data source.

- **Parameters**
  - `source` - the data source to remove (must be same instance)

CLASS **QueryException**

An exception arising from the improper construction of a query. This exception indicates that a query is not complete or not syntactically correct or is missing required constraints.

extends java.lang.Exception

CONSTRUCTORS

`public` **QueryException(** `String` **detailMessage)**

- **Usage**
  - Creates a new Query Exception
- **Parameters**
  - `detailMessage` - a description of the problem with the query

CLASS **ResultsConstraint**

Constrains the query results. For example, specifies the maximum number of results to be returned.

extends java.lang.Object
implements Constraint, Operator

CONSTRUCTORS

`public` **ResultsConstraint(** `int` **shapeType)**

- **Usage**
  - Constraints on the results such as type of geometry and number of results.
- **Parameters**

  – `shapeType` -

`public` **ResultsConstraint**`( int` **shapeType**`, float` **scale** `)`

`public` **ResultsConstraint**`( int` **shapeType**`, float` **scale**`, int` **maxResults**`, boolean` **retrieveGeometry** `)`

Methods

`public int` **getMaxResults**`( )`

- **Usage**
  - The maximum number of results to be returned by query
- **Returns** - The maximum number of results to be returned by query

`public int` **getOperator**`( )`

`public float` **getScale**`( )`

- **Usage**
  - The scale at which spatial data should be retrieved
- **Returns** - the scale to retrieve geometry at

`public int` **getShapeType**`( )`

- **Usage**
  - The type of geometry to retrieve
- **Returns** - The type of geometry to retrieve

`public boolean` **retrieveGeometry**`( )`

- **Usage**
  - Test to see if geometry is to be retrieved. Querys may return just IndexEntrys
- **Returns** - true if geometry is to be retrieved

`public void` **setMaxResults**`( int` **i** `)`

- **Usage**
  - Sets the maximum number of results to be returned
- **Parameters**
  - `i` - the maximum number of results to be returned

`public void` **setOperator**`( int` **operator** `)`

`public void` **setScale**`( float` **f** `)`

- **Usage**
  - Sets the scale at which to retrieve geometry
- **Parameters**
  - `f` - the scale at which to retrieve geometry

`public void` **setShapeType**`( int` **i** `)`

- **Usage**
  - Sets the type of geometry to retrieve. See `ShapeConstants`
- **Parameters**
  - `i` - the type of geometry to retrieve

Class **SpatialConstraint**

A spatial area constraint limits the results to specified physical bounds.

extends java.lang.Object

implements Constraint, Operator

Constructors

`public` **SpatialConstraint**`( Rectangle` **area** `)`

- **Usage**
  - Default constructor

METHODS

`public Rectangle` **getArea**`( )`

- **Usage**
  - The area of this spatial constraint
- **Returns** - the area geometry si being queried for

`public int` **getOperator**`( )`

`public void` **setArea**`( Rectangle` **box** `)`

- **Usage**
  - The area of this spatial constraint
- **Parameters**
  - `box` -

`public void` **setOperator**`( int` **operator** `)`

CLASS **SpatialQuery**

The spatial query interface for constructing queries that can be executed by the framework to retrieve geometry from data sources for display on map layers.

extends java.lang.Object
implements Query, java.lang.Cloneable

CONSTRUCTORS

`public` **SpatialQuery**`( SpatialConstraint` **constraint** `)`

- **Parameters**
  - `constraint` -

Methods

`public void` **addConstraint**`( Constraint` **c** `)`

`public boolean` **constainsConstraint**`( Class` **constraintType** `)`

`public Rectangle` **getArea**`( )`

- **Usage**
  - The area of this spatial constraint
- **Returns** - the area being queried for

`public Constraint` **getConstraint**`( Class` **constraintType** `)`

`public float` **getScale**`( )`

- **Usage**
  - The scale this results constraint
- **Returns** - the scale geometry is being queried for

`public int` **getShapeType**`( )`

- **Usage**
  - The type of geometry data that is to be returned
- **Returns** - the type of geometry being queried for

`public boolean` **retrieveGeometry**`( )`

- **Usage**
  - Indicates whether geometry should be retrieved or just the shape record and its attributes. Used for annotation. Default is true.
- **Returns** - true if geometry is to be returned

D.16 Package ie.tcd.cs.dsg.hermes.gis.index

Package Contents *Page*

Interfaces

INTERFACE **Index**

Interface to Index data. Is extended by specific index types

implements ie.tcd.cs.dsg.hermes.gis.tools.benchmark.IOAccounting

METHODS

`public boolean` **`isEmpty`**`( )`

- **Usage**
  - Tests to see if the index has records in it. Used to find empty index's
- **Returns** - true if there are no entries in the index

INTERFACE **ShapeIndex**

An interface to code indexing ESRI Shapefile shape data

implements Index

Methods

`public boolean` **containsRecord**`(` `int` **recordNumber** `)`

- **Usage**
  - Tests to see if the record is in the index
- **Parameters**
  - `recordNumber` - the record identifier
- **Returns** - true if the record is in the index

`public IndexEntry` **getIndex**`( )`

- **Usage**
  - Gets the index data from the file.
- **Returns** - the index from disk or cache.

`public void` **getRecordOffsets**`(` `ShapeList` **list** `)`

- **Usage**
  - Uses the index to refresh the offsets and content lengths for each of the record numbers specified in the IndexRecord's of each ShapeRecord in the list. This method is used to correct the offsets to geometry in caches (which may not include all the records in the original file).

    If records do not appear at all in the index file then they are removed from the list in order to prevent geometry being read for shapes not in the file.
- **Parameters**
  - `list` - a list of shape records, each with a index record

`public void` **insert**`(` `IndexEntry` **entry** `)`

- **Usage**
  - Inserts a single IndexEntry into the index.
- **Parameters**

  – `entry` - the entry to insert

`public void` **insert**`( ShapeList` **records** `)`

- **Usage**
  – Adds an index entry to this index for each of the shape records specified. Note. Assumes there are no duplicates in the list
- **Parameters**
  – `records` - a list of shape records

`public void` **setScale**`( ScaleRange` **scale** `)`

- **Usage**
  – Slave Shapefiles have scales and ShapeIndex's are used by these files to re-map shape record index offsets. The scale here should match the scale of the .SHP file.
- **Parameters**
  – `scale` - the scale range for this index

`public void` **writeIndex**`( IndexEntry []` **index** `)`

- **Usage**
  – Writes the specified set of index records to this file.
- **Parameters**
  – `index` - the index records to write to the file

Classes

Class **IndexEntry**

Models the Shapefile index file record (SHX).

extends java.lang.Object

implements java.lang.Cloneable

Fields

- public static final int RECORD_LENGTH
  - Number of bytes index entry data takes up on disk (less bounds and id)
- public int offset
  - The offset of a record in the main file is the number of 16-bit words from the start of the main file to the first byte of the record header for the record.
- public int contentLength
  - The content length stored in the index record is the same as the value stored in the main file record header.
- public int id
  - The Shape record number
- public int shapeType
  - The shape type
- public Rectangle bounds
  - The minimum bounding rectangle for shape
- public float scale
  - The scale for this record (Supports multi-scale data sources)
- public long pointOffset
  - Point offset used by R-Tree index's

Constructors

`public` **IndexEntry**`( int` **recNo**`, int` **shpType**`, int` **offset**`, int` **contentLength**`, Rectangle` **mbr** `)`

`public` **IndexEntry**`( Rectangle` **mbr**`, long` **pointOffset** `)`

- **Usage**
  - Constructor. Used by R-Trees for index entries that don't have any data.

METHODS

`public void` **setBounds**`( Rectangle` **box** `)`

- **Parameters**
  - `box` -

`public void` **setListener**`( EntryBoundsChangeListener` **listener** `)`

- **Parameters**
  - `listener` -

D.17 Package ie.tcd.cs.dsg.hermes.gis.ui.event

Package Contents *Page*

Classes

Class MapCentering

Causes the map to center itself if the dot on the contextlayer moves off the screen.

extends java.lang.Object

implements ie.tcd.cs.dsg.hermes.gis.event.LayerListener

Constructors

`public` **MapCentering**`( MapContext` **map**`, DynamicMapLayer` **layer** `)`

- **Parameters**
  - `map` - the map context
  - `layer` - the layer to center the map on

Methods

`public void` **layerChanged**`( LayerEvent` **e** `)`

D.18 Package ie.tcd.cs.dsg.hermes.gis.generalisation.clipping

Package Contents *Page*

Interfaces

Interface **Line2DClip**

Traditionally, polygon clipping has been used to clip out the portions of a polygon that lie outside the window of the output device to prevent undesirable effects.
This interface allows different algorithm for performing clipping against lines to be implemented.

implements ie.tcd.cs.dsg.hermes.gis.tools.Algorithm

Methods

```
public Polyline clipLine( Point  start, Point  end, Rectangle  clip )
```

- **Usage**

  – Clips the line defined by its start and end points

- **Parameters**
  - `start` - the start point of the line
  - `end` - the end point of th eline
  - `clip` - the rectangle to use to clip the line
- **Returns** - a line clipped by the rectangle or null

`public Polyline` **clipLine**`( Polyline` **line**`, Rectangle` **clip** `)`

- **Usage**
  - Cohen-Sutherland Line Clipping
- **Parameters**
  - `line` - a line to clip
  - `clip` - the rectangle to clip by
- **Returns** - null if no clip is supplied otherwise the cliped line

Interface **Polygon2DClip**

An interface to polygon clipping algorithms to allow more than one be implemented.

implements ie.tcd.cs.dsg.hermes.gis.tools.Algorithm

Methods

`public Polygon` **clipPoly**`( Polygon` **poly**`, Rectangle` **clip** `)`

- **Usage**
  - Clips the specified polygon to the specified rectangular bounding box.
- **Parameters**
  - `poly` - the polygon to clip
  - `clip` - the rectangle to clip by
- **Returns** - a clipped polygon

Interface **Polyline2DClip**

An interface for polyline clipping algorithms.

implements Line2DClip

Methods

`public Polyline` **clipPoly**`( Polyline` **lines**`, Rectangle` **clip** `)`

Classes

Class **CohenSutherland**

Repeated clipping is expensive Best used when trivial acceptance and rejection is possible for most lines.
Taken from Computer Graphics for Java Programmers by Leen Ammeraal (1998, ISBN 0-471-98142-7) and altered to support my geometry objects.

extends java.lang.Object
implements Line2DClip, ie.tcd.cs.dsg.hermes.gis.tools.Algorithm

Constructors

`public` **CohenSutherland**`( )`

Methods

`public Polyline` **clipLine**`( Point` **s**`, Point` **e**`, Rectangle` **clip** `)`

`public Polyline` **clipLine**`( Polyline` **line**`, Rectangle` **clip** `)`

`public String` **getName**`( )`

Class **LiangBarsky**

Ported from implementation by Grishul Eugeny: http://en.wikipedia.org/wiki/Liang-Barsky

extends java.lang.Object

implements Line2DClip, ie.tcd.cs.dsg.hermes.gis.tools.Algorithm

Constructors

`public` **LiangBarsky**`( )`

Methods

`public Polyline` **clipLine**`( Point` **s**`, Point` **e**`, Rectangle` **clip** `)`

`public Polyline` **clipLine**`( Polyline` **line**`, Rectangle` **clip** `)`

`public String` **getName**`( )`

Class **PolylineClipper**

Clips polylines. Used to limit the amount of road network data drawn off screen.

extends java.lang.Object

implements Polyline2DClip

Constructors

`public` **PolylineClipper**`( Line2DClip` **lineClipper** `)`

- **Usage**
  - Constructs a Polyline clipper that will clip individual lines using the supplied Line clipping algorithm.

Methods

`public Polyline` **clipLine**`( Point` **start**`, Point` **end**`, Rectangle` **clip** `)`

`public Polyline` **clipLine**`( Polyline` **line**`, Rectangle` **clip** `)`

`public Polyline` **clipPoly**`( Polyline` **lines**`, Rectangle` **clip** `)` `public String` **getName**`( )`

Class **ShapeClip**

Class is responsible for clipping all types of shapes. Delegates clipping to instance of appropriate algorithm implementation as specified by the constructors parameters. These parameters are read directly from the properties file.

extends java.lang.Object
implements ie.tcd.cs.dsg.hermes.gis.tools.Algorithm

Fields

- public static final String LINE_CLIPPER_PROPERTY
  - Polyline clipping algorithm
- public static final String POLY_CLIPPER_PROPERTY
  - Polygon clipping algorithm
- public static final String CLIP_PROPERTY
  - Whether clipping should be used
- public static final String CLIPPER_POINTS_PROPERTY
  - Clipping is expensive so shapes with less than X points don't clip
- public static final String CLIPPER_AREA_PROPERTY
  - If a shape has X times the area of the current view it should be clipped.

Constructors

`public` **ShapeClip**`( String` **lineClipperClass**`, String` **polyClipperClass** `)`

- **Usage**

  - Constructor. Loads specified instances of clipping algorithms.

    Parameters can be passed as nulls if only lines or polys are to be clipped with this instance.

- **Parameters**
  - `lineClipperClass` - the name of a class implementing Line2DClip interface
  - `polyClipperClass` - the name of a class implementing Polygon2DClip interface

METHODS

`public Shape` **clip**`( int` **shpType**`, ShapeRecord` **rec**`, int` **index**`, Rectangle` **clip** `)`

- **Usage**
  - Clips the specified shape to the specified rectangle.
- **Parameters**
  - `shpType` - the type of shape (as read from its record)
  - `rec` - a Shape object
  - `index` - the shape part index
  - `clip` - the rectangle to clip the shape to
- **Returns** - a clipped shape

`public String` **getDescription**`( )`

`public String` **getName**`( )`

CLASS SutherlandHodgman

An implementation of the Sutherland-Hodgman polygon clipping algorithm. This algorithm is restricted to: 1) polygons that are clockwise oriented; 2) polygons can not have holes; and 3)polygons can not be self-intersecting.

See Section 4.4 of Ammeraal, L. (1998) Computer Graphics for Java Programmers, Chichester: John Wiley, ISBN 0-471-98142-7

extends java.lang.Object

implements Polygon2DClip, ie.tcd.cs.dsg.hermes.gis.tools.Algorithm

Constructors

public **SutherlandHodgman**( )

Methods

public Polygon **clipPoly**(Polygon **poly**, Rectangle **clip**)

public String **getName**( )

D.19 Package ie.tcd.cs.dsg.hermes.gis.ui.style

Package Contents *Page*

Interfaces

INTERFACE **Style**

An interface to access attributes that define how graphics are rendered to the screen.

implements java.lang.Cloneable

METHODS

`public Color` **getBackgroundColor**`( )`

- **Usage**
  - Returns the background colour of the drawing canvas
- **Returns** - the background colout of the drawing

`public Color` **getFillColor**`( )`

- **Usage**
  - Returns the colour to fill graphics with when they are rendered to the screen.
- **Returns** - the colour to render graphics

`public Color` **getLineColor**`( )`

- **Usage**
  - Returns the colour that should be used to render lines to the screen.

- **Returns** - the colour to render lines

`public int` **getLineWidth**`( )`

- **Usage**
  - Returns the width of lines.
- **Returns** - the width of lines on the screen in pixels

`public Color` **getTextColor**`( )`

- **Usage**
  - Returns the color that text shoudl be painted
- **Returns** - the color that text shoudl be painted

`public boolean` **isDashedLine**`( )`

- **Usage**
  - Gets the attribute describing whether lines should be dashed. If they are not dashed they are solid.
- **Returns** - true if the line is to be dashed

`public boolean` **isFilled**`( )`

- **Usage**
  - Indicates whether the graphic should be filled when it is rendered.
- **Returns** - true if the drawing is to be filled

`public void` **setFillColor**`( Color` **c** `)`

- **Usage**
  - Sets the colour to use when rendering filled graphics. If the graphic is not to be filled this attribute will be ignored.
- **Parameters**
  - `c` - the colour to fill the graphic with

`public void` **setFilled**`( boolean` **b** `)`

- **Usage**
  - Sets the graphic to appear filled or outlined when it is rendered.
- **Parameters**
  - `b` - whether the graphic is to be filled

`public void` **setLineColor**`( Color` **c** `)`

- **Usage**
  - Sets the colour to render the line on the screen
- **Parameters**
  - `c` - the colour to render the line

`public void` **setTextColor**`( Color` **c** `)`

- **Usage**
  - Sets the color text should be drawn
- **Parameters**
  - `c` - the color text should be drawn

Bibliography

[1] Beyond the world summit on the information society. World Information Society Report, United Nations Conference on Trade and Development 2, International Telecommunication Union, Geneva, May 2007. `http://www.itu.int/osg/spu/publications/worldinformationsociety/2007/WISR07-summary.pdf`.

[2] Gregory Abowd, Christopher Atkeson, Jason Hon, Sue Long, Rob Kooper, and Mike Pinkerton. Cyberguide: A Mobile Context-Aware Tour Guide. *ACM Wireless Networks*, 3:421–433, 1997.

[3] acbPocketSoft . acbTaskMan. Online, October 2008. `http://www.acbpocketsoft.com/Products/acbTaskMan/acbTaskMan-Overview-7.html`.

[4] Maneesh Agrawala and Chris Stolte. Rendering effective route maps: improving usability through generalization. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 241–249, New York, NY, USA, 2001. ACM Press.

[5] John M. Airey, John H. Rohlf, and Jr. Frederick P. Brooks. Towards image realism with interactive update rates in complex virtual building environments. *SIGGRAPH Comput. Graph.*, 24(2):41–50, 1990.

[6] Jochen H. Albrecht. Universal gis operations for environmental modeling. In *Third International Conference/Workshop on Integrating GIS and Environmental Modeling*, Santa Fe, New Mexico, USA, January 1996. CD-ROM.

[7] W. Alsalih, S. Akl, and H. Hassancin. Energy-aware task scheduling: towards

enabling mobile computing over manets. *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, page 8, April 2005.

[8] Brian Amedro, Vladimir Bodnartchouk, Denis Caromel, Christian Delbé, Fabrice Huet, and Guillermo L. Taboada. Current state of java for hpc. Technical Report 0353, Institut National de Recherche en Informatique et en Automatique (INRIA), August 2008. `http://hal.inria.fr/docs/00/31/20/39/PDF/RT-0353.pdf`.

[9] Leen Ammeraal. *Computer Graphics for Java Programmers.* Wiley, April 1998.

[10] Éamonn Linehan and Mike Spence. The hermes framework user's manual. Online, July 2007. `https://www.dsg.cs.tcd.ie/~linehane/hermes/HermesFramework.pdf`.

[11] Arjun Anand, Constantine Manikopoulos, Quentin Jones, and Cristian Borcea. A quantitative analysis of power consumption for location-aware applications on smart phones. In *Proceedings of the IEEE International Symposium on Industrial Electronics*, 2007.

[12] Apple Inc. Maps with gps. Online, October 2008. `http://www.apple.com/iphone/features/maps.html`.

[13] Masatoshi Arikawa, Shin'ichi Konomi, and Keisuke Ohnishi. Navitime: Supporting pedestrian navigation in the real world. *IEEE Pervasive Computing*, 6(3):21–29, 2007.

[14] Marcel Arrufat, Gerard París, and Pedro García López. Agora: an integrated approach for collaboration in manets. In *MOBILWARE '08: Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, pages 1–6, Innsbruck, Austria, February 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[15] Matthias Bader, Mathieu Barrault, Nicolas Regnauld, Sébastien Mustiére, Cêcile Duchêne, Anne Ruas, Emmanuel Fritsch, Francois Lecordix, and Emmanuel Barillot. Agent project, state of the art and selection of basic

algorithms. Technical Report D2, Department of Geography, University of Zurich, February 1999. http://agent.ign.fr/deliverable/DD2.pdf.

[16] Jakob E. Bardram. From Desktop Task Management to Ubiquitous Activity-Based Computing. In Victor Kaptelinin and Mary Czerwinski, editors, *Integrated Digital Work Environments: Beyond the Desktop Metaphor*, pages 49–78. MIT Press, 2007.

[17] Brian A. Barsky. A new concept and method for line clipping. *ACM Trans. Graph.*, 3(1):1–22, 1984.

[18] Matthias Basler and Adrian Custer. Geotools - map, layer, data and rendering architecture. Online, July 2006. http://docs.codehaus.org/display/GEOTOOLS/MapLayerArchitecture+-+4_Progress.

[19] Jörg Baus, Antonio Krüger, and Wolfgang Wahlster. A resource-adaptive mobile navigation system. In *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*, pages 15–22, New York, NY, USA, 2002. ACM.

[20] Rudolf Bayer. Binary b-trees for virtual memory. In *ACM-SIGFIDET Workshop, Session 5B*, pages 219–235, San Diego, California, 1971.

[21] BBN Technologies. Openmap - open systems mapping technology. Online, February 2008. http://openmap.bbn.com/.

[22] Ashweeni Beeharee and Anthony Steed. Filtering location-based information using visibility. *Location and Context-Awareness*, pages 306–315, 2005.

[23] Ashweeni Beeharee and Anthony Steed. Exploiting real world knowledge in ubiquitous applications. *Personal and Ubiquitous Computing*, 11(6):429–437, 2007.

[24] Ashweeni Beeharee and Anthony Steed. *Map-based Mobile Services Design, Interaction and Usability*, chapter 14 Geographical Data in Mobile Applications Uses beyond Map Making, pages 293–309. Lecture Notes in Geoinformation and Cartography. Springer Berlin Heidelberg, 2008.

[25] Petros Belimpasakis, Juha-Pekka Luoma, and Mihaly Börzsei. Content sharing middleware for mobile devices. In *MOBILWARE '08: Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, pages 1–8, Innsbruck, Austria, February 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[26] Steve Benford, Andy Crabtree, Martin Flintham, Adam Drozd, Rob Anastasi, Mark Paxton, Nick Tandavanitj, Matt Adams, and Ju Row-Farr. Can you see me now? *ACM Trans. Comput.-Hum. Interact.*, 13(1):100–133, 2006.

[27] Joseph K. Berry. *Spatial Reasoning for Effective GIS.* John Wiley Publishers, September 1996.

[28] Michela Bertolotto and Max J. Egenhofer. Progressive transmission of vector map data over the world wide web. *Geoinformatica*, 5(4):345–373, 2001.

[29] Jim Blinn. A trip down the graphics pipeline: Line clipping. *IEEE Computer Graphics and Applications*, 11(1):98–105, 1991.

[30] Matthew Bloch and Mark Harrower. Mapshaper. Online, September 2008. `http://www.mapshaper.org/`.

[31] Gaetano Borriello, Matthew Chalmers, Anthony LaMarca, and Paddy Nixon. Delivering real-world ubiquitous location systems. *Communications of the ACM*, 48(3):36–41, 2005.

[32] Jan Bosch, Peter Molin, Michael Mattsson, and PerOlof Bengtsson. Object-oriented framework-based software development: problems and experiences. *ACM Computing Surveys*, 32(1es), March 2000. Article No. 3.

[33] Frantisek Brabec and Hanan Samet. Client-based spatial browsing on the world wide web. *IEEE Internet Computing*, 11(1):52–59, 2007.

[34] Peter J. Brown. The stick-e document: A framework for creating context-aware applications. In *In Proceedings of Electronic Publishing*, volume 8, pages 259–272, June – September 1996.

[35] Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern, and Steven A. Shafer. Easyliving: Technologies for intelligent environments. In *HUC '00: Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing*, pages 12–29, London, UK, 2000. Springer-Verlag.

[36] Barry Brumitt and Steven Shafer. Better living through geometry. *Personal Ubiquitous Computing*, 5(1):42–45, 2001.

[37] Stefano Burigat and Luca Chittaro. *Geographic Data Visualization on Mobile Devices for User's Navigation and Decision Support Activities*, pages 261–284. Springer, 1 edition, September 2007.

[38] Jenna Burrell and Geri K. Gay. E-graffiti: evaluating real-world use of a context-aware system. *Interacting with Computers*, 14(4):301–312, July 2002.

[39] Edwin Earl Catmull. *A subdivision algorithm for computer display of curved surfaces.* Ph.d thesis, The University of Utah, UT, USA, 1974.

[40] Dan Chalmers, Morris Sloman, and Naranker Dulay. Map adaptation for users of mobile systems. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 735–744, New York, NY, USA, 2001. ACM Press.

[41] Guangyu Chen, Byung-Tae Kang, Mahmut Kandemir, Narayanan Vijaykrishnan, Mary Jane Irwin, and Rajarathnam Chandramouli. Studying energy trade offs in offloading computation/compilation in java-enabled mobile devices. *Parallel and Distributed Systems, IEEE Transactions on*, 15(9):795–809, September 2004.

[42] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dartmouth College, Hanover, NH, USA, 2000.

[43] Keith Cheverst, Nigel Davies, Keith Mitchell, and Adrian Friday. Experiences of developing and deploying a context-aware tourist guide: the guide project. In *MobiCom '00: Proceedings of the 6th annual international conference on*

Mobile computing and networking, pages 20–31, New York, NY, USA, 2000. ACM Press.

[44] Keith Cheverst, Keith Mitchell, and Nigel Davies. Design of an object model for a context sensitive tourist guide. *Computers & Graphics*, 23(6):883–891, December 1999.

[45] N. M. Mosharaf Kabir Chowdhury, Md. Mostofa Akbar, and Mohammad Kaykobad. Disktrie: An efficient data structure using flash memory for mobile devices. In M. Kaykobad and Md. Saidur Rahman, editors, *WALCOM*, pages 76–87. Bangladesh Academy of Sciences (BAS), 2007.

[46] Keith C. Clarke. Mobile mapping and geographic information systems. *Cartography and Geographic Information Science*, 31:131–136, July 2004.

[47] Daniel Cohen-Or, Yiorgos L. Chrysanthou, Claudio T. Silva, and Durand Durand. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics*, 09(3):412–431, 2003.

[48] Peter H. Dana. Map projection overview. Online. Revised: October 2000, August 2008. /urlhttp://www.colorado.edu/geography/gcraft/notes/mapproj/mapproj.html.

[49] dataBased Intelligence Inc. dBase about us. Online, October 2008. `http://www.dbase.com/About_us.asp`.

[50] Nigel Davies, Keith Cheverst, Keith Mitchell, and Alon Efrat. Using and determining location in a context-sensitive tour guide. *IEEE Computer*, 34(8):35–41, 2001.

[51] Nigel Davies, Keith Cheverst, Keith Mitchell, and Adrian Friday. 'caches in the air': disseminating tourist information in the guide system. *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, pages 11–19, February 1999.

[52] Ioannis Delikostidis, Corné P.J.M. van Elzakker, and Peter J.M. van Oosterom. Usability testing dynamic maps : overcoming limitations of mobile devices. *The global magazine for geomatics*, 21(12):16–19, 2007.

[53] Giuliana Dettori and Enrico Puppo. Designing a library to support model-oriented generalization. In *GIS '98: Proceedings of the 6th ACM international symposium on Advances in geographic information systems*, pages 34–39, New York, NY, USA, 1998. ACM.

[54] Thomas Devogele, Jenny Trevisan, and Laurent Raynal. Building a multiscale database with scale-transition relationships. In *7th Int. Symposium on Spatial Data Handling, Advances in GIS Research*, pages 6.19–6.33. Delft, 1996.

[55] Anind K. Dey. Understanding and using context. *Personal Ubiquitous Computing*, 5(1):4–7, 2001.

[56] Anind K. Dey and Gregory D. Abowd. CyberMinder: A Context-Aware System for Supporting Reminders. In *Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing (HUC '00)*, pages 172–186, London, UK, 2000. Springer-Verlag.

[57] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A context-based infrastructure for smart environments. In *1st International Workshop on Managing Interactions in Smart Environments*, pages 114–128, 1999.

[58] Anind K. Dey, Daniel Salber, and Gregory D. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16 (2-4):97–16, 2001.

[59] Anind K. Dey, Daniel Salber, Gregory D. Abowd, and Masayasu Futakawa. The conference assistant: Combining context-awareness with wearable computing. In *ISWC '99: Proceedings of the 3rd IEEE International Symposium on Wearable Computers*, page 21, Washington, DC, USA, 1999. IEEE Computer Society.

[60] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[61] Susan E. Dorward. A survey of object-space hidden surface removal. *International Journal of Computational Geometry and Applications*, 4(3):325–362, 1994.

[62] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Canadian Cartographer*, 10:112–122, 1973.

[63] Cormac Driver. *An Application Framework for Mobile, Context-Aware Trails.* PhD thesis, Dept. of Computer Science, Trinity College Dublin, April 2007.

[64] Cormac Driver, Eamonn Linehan, and Siobhán Clarke. A framework for mobile, context-aware trails-based applications: Experiences with an application-led approach. In *Workshop 1 ("What Makes for Good Application-led Research in Ubiquitous Computing?"), Pervasive 05*, 2005.

[65] Max Egenhofer. Spatial information appliances: A next generation of geographic information systems. October 1999.

[66] Per Enge and Pratap Misra. Special issue on global positioning system. *Proceedings of the IEEE*, 87(1):3–15, January 1999.

[67] Ericsson. Mobile applications with j2me. White Paper, Available Online, July 2001. `http://www.j2meworld.com/magazine/2002-data01/j2mewhitepaper.pdf`.

[68] Fredrik Espinoza, Per Persson, Anna Sandin, Hanna Nyström, Elenor Cacciatore, and Markus Bylund. Geonotes: Social and navigational aspects of location-based information systems. In *UbiComp '01: Proceedings of the 3rd international conference on Ubiquitous Computing*, pages 2–17, London, UK, 2001. Springer-Verlag.

[69] ESRI. *ESRI Shapefile Technical Description.* Environmental Systems Research Institute, Inc., July 1998.

[70] ESRI. Arcgis mobile. Online, August 2008. `http://www.esri.com/software/arcgis/arcgismobile/index.html`.

[71] ESRI GIS and Mapping Software. Mobile gis. Online `http://www.esri.com/software/arcgis/about/mobile_gis.html`, September 2007.

[72] Keith I. Farkas, Jason Flinn, Godmar Back, Dirk Grunwald, and Jennifer M. Anderson. Quantifying the energy consumption of a pocket computer and a java virtual machine. *SIGMETRICS Perform. Eval. Rev.*, 28(1):252–263, 2000.

[73] Mohamed E. Fayad and Douglas C. Schmidt. Object-oriented application frameworks (special issue introduction). *Communications of the ACM*, 40(10):39–42, October 1997.

[74] Raphael A. Finkel and Jon Louis Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, March 1974.

[75] Jason Flinn and M. Satyanarayanan. Managing battery lifetime with energy-aware adaptation. *ACM Trans. Comput. Syst.*, 22(2):137–179, 2004.

[76] B. Flyvbjerg. Five misunderstandings about case study research. *Qualitative Inquiry*, 12(2):219–245, April 2006.

[77] Theodor Foerster, Jantien Stoter, Barend Köbben, and Peter van Oosterom. A generic approach to simplification of geodata for mobile applications. In *The 10th AGILE Conference on GIScience*, Aalborg, Denmark, May 2007.

[78] James Foley, Andries van Dam, Steven Feiner, and John Hughes. *Computer Graphics: Principles and Practice, second edition.* Addison-Wesley Professional, 2nd edition, 1990.

[79] Jean-Michel Follin and Alain Bouju. *An Incremental Strategy for Fast Transmission of Multi-Resolution Data in a Mobile System*, chapter 4, pages 57–79. Springer Berlin Heidelberg, 2008.

[80] George H. Forman and John Zahorjan. The challenges of mobile computing. *IEEE Computer*, 27(4):38–47, April 1994.

[81] Dieter Fritsch and Steffen Volz. Nexus - the mobile gis environment. *Joint First Workshop on Mobile Future and Symposium on Trends in Communications, SympoTIC '03.*, pages 26–28, October 2003.

[82] Dieter Fritsch, Steffen Volz, and Darko Klinec. NEXUS - integrating data and services for mobile users of location based services. *Geo-Informations-Systeme (GIS)*, pages 21–25, March 2006.

[83] Tobias Fritsch, Hartmut Ritter, and Jochen Schiller. Mobile phone gaming (a follow-up survey of the mobile phone gaming sector and its users). *Entertainment Computing - ICEC 2006*, pages 292–297, 2006.

[84] Peter Fröhlich, Rainer Simon, Lynne Baillie, and Hermann Anegg. Comparing conceptual designs for mobile access to geo-spatial information. In *MobileHCI '06: Proceedings of the 8th conference on Human-computer interaction with mobile devices and services*, pages 109–112, New York, NY, USA, 2006. ACM.

[85] Peter Fröhlich, Rainer Simon, Lynne Baillie, Joi Roberts, and Roderick Murray-Smith. Mobile spatial interaction. In *CHI '07: CHI '07 extended abstracts on Human factors in computing systems*, pages 2841–2844, New York, NY, USA, 2007. ACM.

[86] Eran Gal and Sivan Toledo. Algorithms and data structures for flash memories. *ACM Comput. Surv.*, 37(2):138–163, 2005.

[87] Keith Gardiner and James D. Carswell. Viewer-based directional querying for mobile applications. *International Conference on Web Information Systems Engineering Workshops (WISEW'03)*, 00:83–91, 2003.

[88] Georg Gartner. Location-based mobile pedestrian navigation services - the role of multimedia cartography. In *International Joint Workshop on Ubiquitous, Pervasive and Internet Mapping (UPIMap2004)*, Tokyo, Japan, September 2004.

[89] Georg Gartner and Susanne Uhlirz. Cartographic concepts for realizing a location based umts service: Vienna city guide lol. In *In Proceedings of the Cartographic Conference*, pages 3229–3239, 2001.

[90] Google Code. Android - an open handset alliance project. Online, October 2008. `http://code.google.com/android/`.

[91] Google Inc. Google maps. Online, August 2008. `http://maps.google.com/`.

[92] Stephen Greaves and Peter R. Stopher. A synthesis of gis and activity-based travel-forecasting. *Geographical Systems*, 5:59–89, 1998.

[93] William G. Griswold, Robert Boyer, Steven W. Brown, and Tan Minh Truong. A component architecture for an extensible, highly integrated context-aware computing infrastructure. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 363–372, Washington, DC, USA, 2003. IEEE Computer Society.

[94] Dietmar Grünreich. Computer-assisted generalisation. In *Papers CERCO Cartography Course*. Institut für Angewandte Geodäsie, Frankfurt a. M., 1985.

[95] Selim Gurun. *Modeling, predicting and reducing energy consumption in resource restricted computers*. PhD thesis, Santa Barbara, CA, USA, 2007. Adviser-Chandra Krintz.

[96] Selim Gurun, Priya Nagpurkar, and Ben Y. Zhao. Energy consumption and conservation in mobile peer-to-peer systems. In *MobiShare '06: Proceedings of the 1st international workshop on Decentralized resource sharing in mobile computing and networking*, pages 18–23, New York, NY, USA, 2006. ACM.

[97] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In Beatrice Yormark, editor, *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 47–57. ACM Press, 1984.

[98] Mark Hampe, Monika Sester, and Lars Harrie. Generating and using a multi-representation data-base (mrdb) for mobile applications. In *Papers of the ICA Workshop on Generalisation and Multiple Representation*, Leicester, August 2004.

[99] Mark Hampe, Monika Sester, and Lars Harrie. Multiple representation databases to support visualization on mobile devices. In *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 135–140, Istanbul, Turkey, July 2004.

[100] Qiang Han and Michela Bertolotto. A multi-level data structure for vector maps. In *GIS '04: Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pages 214–221, New York, NY, USA, 2004. ACM Press.

[101] Lars Harrie and Anna-Karin Hellström. A prototype system for propagating updates between cartographic data sets. *The Cartographic Journal*, 36(2):133–140, 1999.

[102] M. Harrower and M. Bloch. Mapshaper.org: a map generalization web service. *Computer Graphics and Applications, IEEE*, 26(4):22–27, July – August 2006.

[103] Karen Henricksen and Jadwiga Indulska. A software engineering framework for context-aware pervasive computing. In *PERCOM '04: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, page 77, Washington, DC, USA, 2004. IEEE Computer Society.

[104] Hewlett-Packard. Hp ipaq hx2490 pocket pc overview, north america, version 7. Online, September 2007. `http://h18000.www1.hp.com/products/quickspecs/12293_na/12293_na.PDF`.

[105] Jason I. Hong and James A. Landay. An infrastructure approach to context-aware computing. *Human-Computer Interaction*, 16(2):287–303, 2001.

[106] Open Geospatial Consortium Inc. Opengis location service (openls) implementation specification: Core services. OpenGIS Implementation Specification 1.1, Open Geospatial Consortium Inc., May 2005.

[107] International Cartographic Association. *Multilingual Dictionary of Technical Terms in Cartography.* Steiner, Wiesbaden, Germany, 1973.

[108] Stephen S. Intille, Kent Larson, J. S. Beaudin, J. Nawyn, E. Munguia Tapia, and P. Kaushik. A living laboratory for the design and evaluation of ubiquitous computing technologies. In *CHI '05 extended abstracts on Human factors in computing systems*, pages 1941–1944, New York, NY, USA, 2005. ACM Press.

[109] Noritaka Ishizumi, Keizo Saisho, and Akira Fukuda. A design of flash memory file system for embedded systems. *Systems and Computers in Japan*, 35(1):91–100, 2004.

[110] Ivar Jacobson, Martin Griss, and Patrik Jonsson. *Software Reuse: Architecture, Process and Organization for Business Success.* ACM Press Books, 1997.

[111] Javid Jamae. Bring java's system.currenttimemillis() back into the fold for transaction monitoring. Online, DevX, Jupitermedia Corporation, July 2005. `http://www.devx.com/Java/Article/28685`.

[112] Tony Jones. Calculating the distance to the horizon. Online, 2005. `http://www.wolfram.demon.co.uk/rp_horizon_distance.html`.

[113] Hae-Kyong Kang and Ki-Joune Li. A framework for dynamic updates of map data in mobile devices. *Web and Wireless Geographical Information Systems*, pages 66–77, 2005.

[114] Cory D. Kidd, Robert Orr, Gregory D. Abowd, Christopher G. Atkeson, Irfan A. Essa, Blair MacIntyre, Elizabeth D. Mynatt, Thad Starner, and Wendy Newstetter. The aware home: A living laboratory for ubiquitous computing research. In *CoBuild '99: Proceedings of the Second International Workshop on Cooperative Buildings, Integrating Information, Organization, and Architecture*, pages 191–198, London, UK, 1999. Springer-Verlag.

[115] A. Kilgour. Unifying vector and polygon algorithms for scan conversion and clipping. In *Eurographics '87*, pages 363–375, Amsterdam, August 1987.

[116] Tiina Kilpelainen. Updating multiple representation geo-databases by incremental generalization. In H. Ebner, C. Heipke, and K. Eder, editors, *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 2357 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 440–447, Munich, Germany, August 1994.

[117] Tiina Kilpelainen. Updating multiple representation geo-databases by incremental generalization. *ISPRS Commission III Symposium: Spatial*

Information from Digital Photogrammetry and Computer Vision, 2357(1):440–447, 1994.

[118] Simon G. M. Koo, Catherine Rosenberg, Hoi ho Chan, and Yat Chung Lee. Location discovery in enterprise-based wireless networks: Implementation and applications. In *In Proceedings of the 2nd IEEE Workshop on Applications and Services in Wireless Networks (ASWN 2002*, pages 3–5, July 2002.

[119] Menno-Jan Kraak. Current trends in visualisation of geospatial data with special reference to cartography. In *Proceedings of the XXIIth INCA Congress*, volume 22 of *Convergence Of Imagery, Information & Maps*, pages 319–324, Ahmedabad, 2002. Indian National Cartographic Association. invited paper.

[120] Christian Kray. *Situated Interaction on Spatial Topics.* Doktor der ingenieurwissenschaften (dr.-ing.), der Naturwissenschaftlich-Technischen Fakultĺat I der Universitĺat des Saarlandes, Saarbrĺucken, May 2003.

[121] Christian Kray, Christian Elting, Katri Laakso, and Volker Coors. Presenting route instructions on mobile devices. In *IUI '03: Proceedings of the 8th international conference on Intelligent user interfaces*, pages 117–124, New York, NY, USA, 2003. ACM Press.

[122] Steinar Kristoffersen and Fredrik Ljungberg. "making place" to make it work: empirical explorations of hci for mobile cscw. In *GROUP '99: Proceedings of the international ACM SIGGROUP conference on Supporting group work*, pages 276–285, New York, NY, USA, 1999. ACM.

[123] Partha Kuchana. *Software Architecture Design Patterns in Java.* AUERBACH, April 2004.

[124] Christoph Clemens Lee. Javancss - a source measurement suite for java. Online, October 2008. `http://www.kclee.de/clemens/java/javancss/`.

[125] Alexander Leonhardi and Martin Bauer. The vit-system: Experiences with developing a location-aware system for the internet. In *HUC2k Workshop on Infrastructure for Smart Devices*, Bristol, UK, September 2000.

[126] J. P. Lewis and Ulrich Neumann. Performance of java versus c++. Online, September 2004.
`http://www.idiom.com/~zilla/Computer/javaCbenchmark.html`.

[127] Zhilin Li. Digital map generalization at the age of enlightenment: a review of the first forty years. *Cartographic Journal, The*, 44:80–93(14), February 2007.

[128] Zhiyuan Li, Cheng Wang, and Rong Xu. Computation offloading to save energy on handheld devices: a partition scheme. In *CASES '01: Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 238–246, New York, NY, USA, 2001. ACM.

[129] Sue Long, Rob Kooper, Gregory D. Abowd, and Christopher G. Atkeson. Rapid prototyping of mobile context-aware applications: The cyberguide case study. In *MobiCom '96: Proceedings of the 2nd annual international conference on Mobile computing and networking*, pages 97–107, New York, NY, USA, 1996. ACM Press.

[130] Paul A. Longley, Michael F. Goodchild, David J. Maguire, and David W. Rhind. *Geographic Information Systems and Science*. John Wiley & Sons, April 2005.

[131] William A. Mackaness, Anne Ruas, and L. Tiina Sarjakoski., editors. *Generalisation of geographic modelling and application : cartographic modelling and applications*. Elsevier B. V., Boston, MA, April 2007.

[132] Stefan Maierhofer, Rainer Simon, and Robert F. Tobler. Simplified guided visibility sampling for location based services. In *12th International Conference on Urban Planning and Regional Development in the Information Society, Geo Multimedia 007, 2nd Vienna Real Estate Conference (CORP 2007)*, Vienna, Austria, May 2007.

[133] Patrick-Gilles Maillot. A new, fast method for 2d polygon clipping: analysis and software implementation. *ACM Trans. Graph.*, 11(3):276–290, 1992.

[134] Rainer Malaka and Alexander Zipf. Deep map - challenging it research in the

framework of a tourist information system. *Information and communication technologies in tourism 2000*, pages 15–27, 2000.

[135] MapQuest, Inc. Mapquest. Online, September 2008. `http://www.mapquest.com/beta`.

[136] Natalia Marmasse and Chris Schmandt. Location-aware information delivery with commotion. In *HUC '00: Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing*, pages 157–171, London, UK, 2000. Springer-Verlag.

[137] Thomas L. Martin, Daniel P. Siewiorek, Asim Smailagic, Matthew Bosworth, Matthew Ettus, and Jolin Warren. A case study of a system-level approach to power-aware computing. *Trans. on Embedded Computing Sys.*, 2(3):255–276, 2003.

[138] Atsushi Maruyama, Naoki Shibata, Yoshihiro Murata, Keiichi Yasumoto, and Minoru Ito. P-Tour: A Personal Navigation System for Tourism. In *Proceedings of the 11th World Congress on Intelligent Transport Systems*, volume 2, pages 18–21, 2004.

[139] Andrew J. May, Tracy Ross, Steven H. Bayer, and Mikko J. Tarkiainen. Pedestrian navigation aids: information requirements and design implications. *Personal Ubiquitous Computing*, 7(6):331–338, 2003.

[140] Ted McFadden and Jadwiga Indulska. Context-aware environments for independent living. In M. Underwood and K. Suridge, editors, *Proceedings of the Third National Conference for Emerging Researchers in Ageing*, pages 147–151, Brisbane , Australia, December 2004. The University of Queensland.

[141] Robert B. McMaster. A mathematical evaluation of simplification algorithms. *Auto Carto 6 Proceedings*, 2:267–276, 1983.

[142] Liqiu Meng. Towards individualization of mapmaking and mobility of map use. In *Proceedings of the 20th International Cartographic Conference*, volume 1, pages 60–68, Beijing, 2001.

[143] Liqiu Meng and Tumasch Reichenbacher. *Map-based Mobile Services - Theories, Methods and Implementations*, chapter 1. Map based Mobile Services, pages 1–10. Springer, 2005.

[144] Liqiu Meng and Tumash Reichenbacher. Geodienste für location based services und geovisualisierungsdienste. *Tagungsband zum 8. Münchner Fortbildungsseminar Geoinformationssysteme*, 12:1–12, 2003.

[145] Microsoft Luxembourg S.Ã. Live search maps. Online, September 2008. `http://maps.live.com/`.

[146] Sun Microsystems. Java Micro Edition: Personal Basis Profile. Online; accessed 26-September-2006. `http://java.sun.com/products/personalprofile/`.

[147] Robert B. Miller. Response Time in Man-Computer Conversational Transactions. In *Fall Joint Computer Conference 33 (part 1)*, pages 267–277. AFIPS Press, 1968.

[148] Alexandra Millonig and Katja Schechtner. Developing landmark-based pedestrian-navigation systems. *Intelligent Transportation Systems, IEEE Transactions on*, 8(1):43–49, March 2007.

[149] Tyler Mitchell. *Web Mapping Illustrated - Using Open Source GIS Toolkits.* O'Reilly, June 2005.

[150] Jean-Claude Müller, Robert Weibel, Jean-Philippe Lagrange, and Francois Salgé. Generalization: state of the art and issues. In J.C. Müller, J.P. Lagrane, and R. Weibel, editors, *GIS and Generalization: Methodology and Practice*, pages 3–17. Taylor and Francis, 1995.

[151] Madan Kumar M.M, Amit Thawani, Sridhar V, and Y. N. Srikant. Analysis of application partitioning for massively multiplayer mobile gaming. In *MOBILWARE '08: Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, pages 1–6, Innsbruck, Austria, February 2008. ICST.

[152] Martin Modahl, Bikash Agarwalla, Scott Saponas, Gregory Abowd, and Umakishore Ramachandran. Ubiqstack: A taxonomy for a ubiquitous computing software stack. volume 10, pages 21–27, London, UK, 2005. Springer-Verlag.

[153] Hossein Mohammadi, Abbas Rajabifard, and Ian Williamson. Spatial data integrability and interoperability in the context of sdi. *The European Information Society*, pages 401–413, 2008.

[154] S. Mohapatra, N. Dutt, A. Nicolau, and N. Venkatasubramanian. Dynamo: A cross-layer framework for end-to-end qos and energy optimization in mobile handheld devices. *Selected Areas in Communications, IEEE Journal on*, 25(4):722–737, May 2007.

[155] Thomas P. Moran and Paul Dourish. Introduction to this special issue on context-aware computing. *Human-Computer Interaction*, 16(2):87–95, 2001.

[156] Multi Media Mapping Ltd., trading as Multimap. Multimap open api v1.2 documentation. Online, September 2008. `http://www.multimap.com/openapidocs/1.2/`.

[157] Richard R. Muntz, editor. *IT Roadmap to a Geospatial Future*. THE NATIONAL ACADEMIES PRESS, Washington, D.C, 2003. Committee on Intersections Between Geospatial Information and Information Technology, National Research Council.

[158] Hani Naguib, George Coulouris, and Scott Mitchell. Middleware support for context-aware multimedia applications. In *DAIS'2001, The Third IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems, Krakov, Poland*, pages 9–22, September 2001.

[159] Fui Hoon Nah and Kihyun Kim. *Managing Web-enabled Technologies in Organizations: a global perspective*, chapter 7, pages 146–161. Idea Group Publishing, Hershey, PA, USA, 2000.

[160] Tina M. Nicholl, D. T. Lee, and Robin A. Nicholl. An efficient new algorithm for 2-d line clipping: Its development and analysis. In *SIGGRAPH '87:*

Proceedings of the 14th annual conference on Computer graphics and interactive techniques, pages 253–262, New York, NY, USA, 1987. ACM Press.

[161] Daniela Nicklas, Matthias GroSSmann, Thomas Schwarz, Steffen Volz, and Bernhard Mitschang. A model-based, open architecture for mobile, spatially aware applications. In *SSTD '01: Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*, pages 117–135, London, UK, 2001. Springer-Verlag.

[162] Daniela Nicklas, Matthias Grossmann, and Thomas Schwarz. Nexusscout: an advanced location-based application on a distributed, open mediation platform. In *vldb'2003: Proceedings of the 29th international conference on Very large data bases*, pages 1089–1092. VLDB Endowment, 2003.

[163] Daniela Nicklas, Matthias Großmann, Thomas Schwarz, and Steffen Volz. Architecture and data model of nexus. Article in journal, Universität Stuttgart : Sonderforschungsbereich SFB 627 (Nexus: Umgebungsmodelle für mobile kontextbezogene Systeme), September 2001.

[164] Daniela Nicklas and Bernhard Mitschang. The nexus augmented world model: An extensible approach for mobile, spatially aware applications. In Yingxu Wang, Shushma Patel, and Ronald Johnston, editors, *OOIS*, pages 392–. Springer, 2001.

[165] Daniela Nicklas and Bernhard Mitschang. On building location-aware applications using an open platform based on the nexus augmented world model. *Software and Systems Modeling*, 3(4):303–313, December 2004.

[166] Jakob Nielsen. *Usability Engineering.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[167] Jakob Nielsen. Response Times: The Three Important Limits. Online, October 2006. `http://www.useit.com/papers/responsetime.html`.

[168] Flemming Nissen, Anders Hvas, Jørgen Münster-Swendsen, and Lars Brodersen. Small-display cartography. Deliverable D3.1.1, Public EC report IST-2000-30090, GiMoDig-project, February 2003.

[169] Annu-Maaria Nivala and L. Tiina Sarjakoski. Need for context-aware topographic maps in mobile devices. In Kirsi Virrantaus and Hsvard Tveite, editors, *ScanGIS'2003 - The 9th Scandinavian Research Conference on Geographical Information Science*, pages 15–29, Espoo, Finland, June 2003. Department of Surveying, Helsinki University of Technology.

[170] Berkeley Institute of Design. Workshop on mobile device applications. UC Berkley, May 2006.

[171] Open GIS Consortium, Inc. Opengiső implementation specification for geographic information - simple feature access - part 1: Common architecture. OpenGISő Implementation Specification 1.2.0, October 2006. `http://www.opengeospatial.org/standards/go`.

[172] Oracle. Leveraging location-based services for mobile applications. Technical white paper, Oracle, 2001.

[173] Mazliza Othman and Stephen Hailes. Power conservation strategy for mobile computers using load sharing. *SIGMOBILE Mob. Comput. Commun. Rev.*, 2(1):44–51, 1998.

[174] Guilhem Paroux, lsabelle Demeure, and Laurent Reynaud. A power-aware middleware for mobile ad-hoc networks. In *NOTERE '08: Proceedings of the 8th international conference on New technologies in distributed systems*, pages 1–7, New York, NY, USA, 2008. ACM.

[175] Jason Pascoe, Nick Ryan, and David Morse. Using while moving: Hci issues in fieldwork environments. *ACM Trans. Comput.-Hum. Interact.*, 7(3):417–437, 2000.

[176] Cynthia A. Patterson, Richard R. Muntz, and Cherri M. Pancake. Challenges in location-aware computing. *IEEE Pervasive Computing*, 02(2):80–89, 2003.

[177] Andrea Piras, Roberto Demontis, Emanuela De Vita, and Stefano Sanna. Compact gml: merging mobile computing and mobile cartography. In *GML And Geo-Spatial Web Services Conference*, Vancouver, British Columbia, July 2004.

[178] Stefan Poslad, Heimo Laamanen, Rainer Malaka, Achim Nick, Phil Buckle, and Alexander Zipf. Crumpet: creation of user-friendly mobile services personalised for tourism. *Second International Conference on 3G Mobile Communication Technologies*, pages 28–32, 2001.

[179] Günther Pospischil, Martina Umlauft, and Elke Michlmayr. Designing lol@, a mobile tourist guide for umts. In *Mobile HCI '02: Proceedings of the 4th International Symposium on Mobile Human-Computer Interaction*, pages 140–154, London, UK, 2002. Springer-Verlag.

[180] Jeffrey S. Poulin. *Measuring software reuse: principles, practices, and economic models.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.

[181] Sham Prasher, Xiaofang Zhou, and Masaru Kitsuregawa. Dynamic multi-resolution spatial object derivation for mobile and www applications. *World Wide Web*, 6(3):305–325, 2003.

[182] Bratislav Predic and Dragan Stojanovic. A framework for handling mobile objects in location based services. In *8th Conference on Geographic Information Science - AGILE 2005*, pages pp. 419–427, Estoril, Lisbon, Portugal, May 2005.

[183] Bratislav Predic, Dragan Stojanovic, and Slobodanka Djordjevic-Kajan. Developing context aware support in mobile gis framework. In *9th Association Geographic Information Laboratories Europe (AGILE) Conference on Geographic Information Science*, pages 90–97, Visegrad, Hungary, May 2006.

[184] Nissanka B. Priyantha, Allen K.L. Miu, Hari Balakrishnan, and Seth Teller. The cricket compass for context-aware mobile applications. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 1–14, New York, NY, USA, 2001. ACM.

[185] Anand Ranganathan, Jalal Al-Muhtadi, Shiva Chetan, Roy Campbell, and M. Dennis Mickunas. Middlewhere: A middleware for location awareness in ubiquitous computing applications. In *Middleware '04: Proceedings of the 5th*

ACM/IFIP/USENIX international conference on Middleware, pages 397–416, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[186] Bill Rankin. Radical cartography / projections / reference. Online, 2006. `http://www.radicalcartography.net/?projectionref`.

[187] Venkat Rao, Gaurav Singhal, Anshul Kumar, and Nicolas Navet. Battery model for embedded systems. *VLSI Design, 2005. 18th International Conference on*, pages 105–110, January 2005.

[188] Siva Ravada. Spatial database services for location-aware applications. Online, Spatial Products Division, Oracle Corporation, 1998. `http://www.gisdevelopment.net/technology/lbs/techlbs004c.htm`.

[189] Tumasch Reichenbacher. The world in your pocket - towards a mobile cartography. In *Proceedings of the 20th International Cartographic Conference*, Beijing, China, 2001.

[190] Tumasch Reichenbacher. *Mobile Cartography - Adaptive Visualisation of Geographic Information on Mobile Devices.* PhD thesis, Department of Cartography, Technical University of Munich, Germany, November 2004.

[191] Tumasch Reichenbacher. Adaptation in mobile and ubiquitous cartography. *Multimedia Cartography*, pages 383–397, 2007.

[192] Josephine Reid, Erik Geelhoed, Richard Hull, Kirsten Cater, and Ben Clayton. Parallel worlds: immersion in location-based experiences. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1733–1736, New York, NY, USA, 2005. ACM.

[193] Jun Rekimoto. Tilting operations for small screen interfaces. In *UIST '96: Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 167–168, New York, NY, USA, 1996. ACM.

[194] Claus Rinner. *Map-based Mobile Services Design, Interaction and Usability*, chapter 16 Mobile Maps and More - Extending Location-Based Services with Multi-Criteria Decision Analysis, pages 335–352. Lecture Notes in Geoinformation and Cartography. Springer Berlin Heidelberg, 2008.

[195] D. Rossi, M. Mellia, and C. Casetti. User patience and the web: a hands-on investigation. *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, 7:4163–4168 vol.7, December 2003.

[196] John C. Russ. *The Image Processing Handbook.* CRC Press, 5 edition, 2007.

[197] Nick S. Ryan, Jason Pascoe, and David R. Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. In V. Gaffney, M. van Leusen, and S. Exxon, editors, *Computer Applications in Archaeology 1997*, British Archaeological Reports, Oxford, October 1998. Tempus Reparatum.

[198] Ivo Salmre. *Writing Mobile Code: Essential Software Engineering for Building Mobile Applications*, chapter Chapter 2: Characteristics of Mobile Applications, pages 19 – 36. Addison-Wesley Professional., 1st edition, February 2005.

[199] Suprateek Sarker and John D. Wells. Understanding mobile handheld device use and adoption. *Communications of the ACM*, 46(12):35–40, 2003.

[200] Ichiro Satoh. A spatial model for ubiquitous computing services. *IEICE Transactions on Communications*, E88-B(3):923–931, 2005.

[201] Ichiro Satoh. Location-based services in ubiquitous computing environments. *International Journal on Digital Libraries*, 6:280–291, June 2006.

[202] Mahadev Satyanarayanan. Fundamental challenges in mobile computing. In *PODC '96: Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, pages 1–7, New York, NY, USA, 1996. ACM.

[203] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US, 1994.

[204] Bill N. Schilit, Norman Adams, Rich Gold, Michael Tso, and Roy Want. The parctab mobile computing system. In *Proceedings Fourth Workshop on Workstation Operating Systems (WWOS-IV)*, pages 34–39. IEEE, October 1993.

[205] Albrecht Schmidt, Michael Beigl, and Hans-W. Gellersen. There is more to context than location. *Computers and Graphics*, 23:893–901(9), December 1999.

[206] Barbara Schmidt-Belz, Mikko Laukkanen, Heimo Laamanen, Manuel Veríssimo, Alex Zipf, Hidir Aras, and Stefan Poslad. Crumpet, user trials and validation results. Report IST-1999-20147, Information Society Technologies (IST), March 2003.

[207] Stefan Schmitz, Alexander Zipf, and Hidir Aras. Open gml-based mobile geo-data-handling for pdas. In *Annual Conf. of the Int. Association for Mathematical Geology (IAMG 2002)*, Berlin, Germany, September 2002.

[208] Monika Sester and Claus Brenner. Continuous generalization for visualization on small mobile devices. In Peter Fisher, editor, *Developments in Spatial Data Handling - 11th International Symposium on Spatial Data Handling*, pages 469–480. Springer Verlag, 2004.

[209] Vidya Setlur, Yingqing Xu, Xuejin Chen, and Bruce Gooch. Retargeting vector animation for small displays. In *MUM '05: Proceedings of the 4th international conference on Mobile and ubiquitous multimedia*, pages 69–77, New York, NY, USA, 2005. ACM Press.

[210] Shashi Shekhar, Mark Coyle, Braiesh Goyal, Duen-Ren Liu, and Shvamsundar Sarkar. Experiences with data models in geographic information systems. *Communications of the ACM*, 40(4), April 1997.

[211] D. P. Siewiorek. Energy locality: Processing/communication/interface tradeoffs to optimize energy in mobile systems. In *WVLSI '01: Proceedings of the IEEE Computer Society Workshop on VLSI 2001*, page 1, Washington, DC, USA, 2001. IEEE Computer Society.

[212] Rainer Simon and Peter Fröhlich. A mobile application framework for the geospatial web. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 381–390, New York, NY, USA, 2007. ACM.

[213] Rainer Simon, Peter Fröhlich, and Hermann Anegg. Enabling spatially aware mobile applications. *Transactions in GIS*, 11(5):783–794, October 2007.

[214] Rodger W. Sinnott. Virtues of the haversine. *Sky and Telescope*, 68(2):159, 1984.

[215] John P. Snyder. Map projections: A working manual. Professional Paper 1395, U.S. Geological Survey, Washington, 1987.

[216] Timothy Sohn, Kevin A. Li, Gunny Lee, Ian E. Smith, James Scott, and William G. Griswold. Place-Its: A Study of Location-Based Reminders on Mobile Phones. In *Proceedings of the 7th International Conference on Ubiquitous Computing (UbiComp 2005)*, Lecture Notes in Computer Science, pages 232–250. Springer, September 2005.

[217] Robert F. Sproull. *Principles of interactive computer graphics (2nd ed.).* McGraw-Hill, Inc., New York, NY, USA, 1979.

[218] Steffen Staab, Hannes Werthner, Francesco Ricci, Alexander Zipf, Ulrike Gretzel, Daniel R. Fesenmaier, Cecile Paris, and Craig Knoblock. Intelligent systems for tourism. *IEEE Intelligent Systems*, 17(6):53–64, 2002.

[219] John Stell and Michael Worboys. Stratified map spaces: A formal basis for multi-resolution spatial databases. In *SDH'98 Proceedings 8th International Symposium on Spatial Data Handling*, pages 180–189. International Geographical, 1998.

[220] Inc. Sun Microsystems. Java tuning white paper. White Paper `http://java.sun.com/performance/reference/whitepapers/tuning.html`, December 2005.

[221] Myles Sutherland and Allan Laframboise. Introduction to the arcgis mobile sdk. Online, 2007. ESRI Virtual Campus.

[222] L. Tiina Sarjakoski Tapani Sarjakoski. The gimodig public final report. Online, March 2005.

[223] Martina Umlauft, Günther Pospischil, Georg Niklfeld, and Elke Michlmayr. Lol@, a mobile tourist guide for umts. *Information Technology & Tourism*, 5:151–164 (14), January 2003.

[224] Miller S. Cartwright W. Urquhart, K. *LBS and TeleCartography*, volume 66, chapter An user-centered approach to designing useful geospatial representations for LBS, pages 69–79. Springer Berlin Heidelberg, 2003.

[225] Jaap van Ekris. What is a good mobile application anyway? Modern Nomads Online Magazine, April 2006. `http://www.modernnomads.info/articles/read.php?article_id=5`.

[226] Chris Veness. Distance between pair of latitude/longitude points. Online, September 2008. Movable Type Ltd `http://www.movable-type.co.uk/scripts/latlong.html`.

[227] Kirsi Virrantaus, Jouni Markkula, Artem Garmash, Vagan Y. Terziyan, Jari Veijalainen, Artem Katasonov, and Henry Tirri. Developing gis-supported location-based services. *Web Information Systems Engineering, 2001. Proceedings of the Second International Conference on*, 2:66–75, December 2001.

[228] Mahes Visvalingam and James D. Whyatt. Line generalisation by repeated elimination of points. *The Cartographic Journal*, 30(1):46–51, 1993.

[229] Steffen Volz, Monika Sester, Dieter Fritsch, and Alexander Leonhardi. Multi-scale data sets in distributed environments. *International Archives of Photogrammetry and Remote Sensing, Part B4, Technical IV/I*, XXXIII, 2000.

[230] Alessandro Cecconi von Vacallo TI. *Integration of Cartographic Generalization and Multi-Scale Databases for Enhanced Web Mapping.* PhD thesis, University of Zürich, Zürich, Switzerland, 2003.

[231] Donggen Wang and Tao Cheng. A spatio-temporal data model for activity-based transport demand modelling. *International Journal of Geographical Information Science*, 15:561–585(25), September 2001.

[232] Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. The active badge location system. *ACM Trans. Inf. Syst.*, 10(1):91–102, 1992.

[233] Andy Ward, Alan Jones, and Andy Hopper. A new location technique for the active office. *Personal Communications, IEEE [see also IEEE Wireless Communications]*, 4(5):42–47, October 1997.

[234] Stefan Weber, Vinny Cahill, Siobhan Clarke, and Mads Haahr. Wireless ad hoc network for dublin: A large-scale ad hoc network test-bed. *ERCIM News*, 54, 2003.

[235] Robert Weibel and Geoffrey H. Dutton. *Geographical Information Systems: Principles and Technical Issues*, chapter Generalising Spatial Data and Dealing with Multiple Representations, pages 125–155. Wiley, New York, US, 2 edition, 1999.

[236] Kevin Weiler and Peter Atherton. Hidden surface removal using polygon area sorting. In *SIGGRAPH '77: Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, pages 214–222, New York, NY, USA, 1977. ACM Press.

[237] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced cpu energy. In *OSDI '94: Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, page 2, Berkeley, CA, USA, 1994. USENIX Association.

[238] Wikipedia, the free encyclopedia. Naked eye. Online, September 2008. `http://en.wikipedia.org/wiki/Naked_eye`.

[239] Stefan Winkler, Karthik Rangaswamy, and ZhiYing Zhou. Intuitive map navigation on mobile devices. *Universal Access in Human-Computer Interaction. Ambient Interaction*, pages 605–614, 2007.

[240] Peter Wonka, Michael Wimmer, Kaichi Zhou, Stefan Maierhofer, Gerd Hesina, and Alexander Reshetov. Guided visibility sampling. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 494–502, New York, NY, USA, 2006. ACM.

[241] Yahoo! Inc. Yahoo! maps, driving directions, and traffic. Online, September 2008. `http://maps.yahoo.com/`.

[242] Huiling Yang. Polygon clipping background theory. Online, April 1998. `http://www.cs.rit.edu/~icss571/clipTrans/PolyClipBack.html`.

[243] Wai yeung Yan. Mobile map service with scalable vector graphics. *Geoscience and Remote Sensing Symposium, 2004. IGARSS '04. Proceedings. 2004 IEEE International*, 5:2967–2970, September 2004.

[244] Hansong Zhang. *Effective occlusion culling for the interactive display of arbitrary models.* PhD thesis, Chapel Hill, NC, USA, 1998.

[245] Alexander Zipf. User-adaptive maps for location-based services for tourism. In Karl W. Wöber, Andrex J. Frew, and Martin Hitz, editors, *Proc. of the 9th Int. Conf. for Information and Communication Technologies in Tourism, ENTER*, Innsbruck, Austria, 2002. Springer Computer Science, Heidelberg, Berlin.

[246] Alexander Zipf. User-adaptive maps for location-based services (lbs) for tourism. In K. Woeber, A. Frew, and M. Hitz, editors, *Proc. of the 9th Int. Conf. for Information and Communication Technologies in Tourism, ENTER 2002*, Innsbruck, Austria, 2002. Springer-Verlag, Heidelberg.