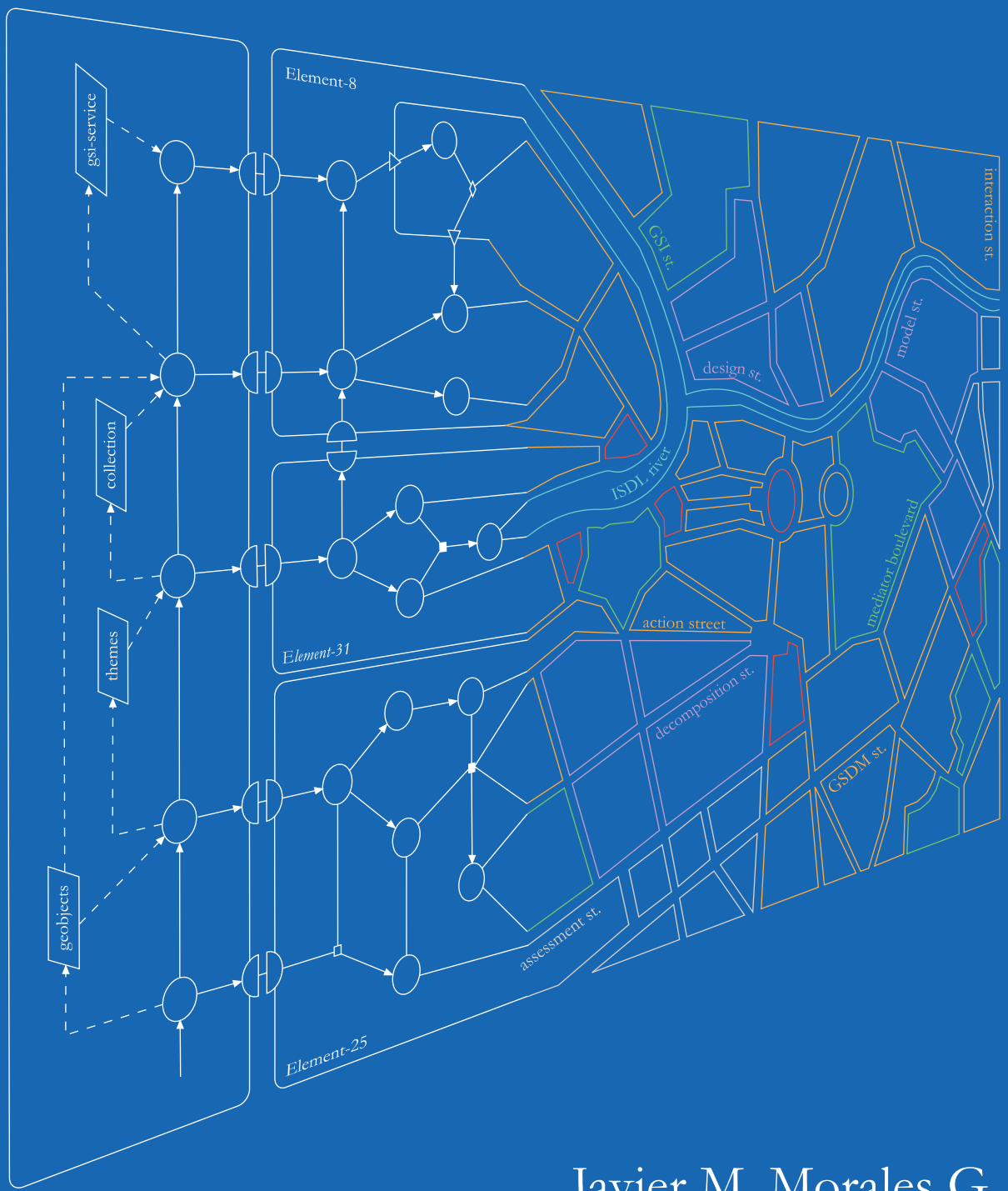
Model-driven Design of Geo-information Services



Javier M. Morales G.

MODEL-DRIVEN DESIGN OF GEO-INFORMATION SERVICES

Javier Marcelino Morales Guarin

March 2004

INTERNATIONAL INSTITUTE FOR GEO-INFORMATION SCIENCE AND EARTH OBSERVATION, ENSCHEDE, THE NETHERLANDS

ITC Dissertation number 110
ITC, P.O. Box 6, 7500 AA Enschede, The Netherlands

CTIT Ph.D.-thesis series, no. 03-61
CTIT, P.O. Box 217, 7500 AE Enschede, The Netherlands

ISSN 1381-3617

ISBN 90-6164-222-1

Printed by ITC Printing Department

MODEL-DRIVEN DESIGN OF GEO-INFORMATION SERVICES

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof.dr. F.A. van Vught,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op vrijdag 26 maart 2004 om 15.00 uur.

door

Javier Marcelino Morales Guarin

geboren op 16 januari 1968
te Bogotá, Colombia

This thesis is approved by
prof. dr. ir. C. A. Vissers, promotor

To the memory
of my Father
Marcelino Morales Bastidas
1938–1988

Preface

The technological advances of recent years have brought many changes to today's society. These changes have impacted the geo-information community, altering the way geo-information is perceived, collected, managed and used. The advent of sophisticated communication infrastructures, like the World Wide Web for example, opened the possibility for geo-information users to exchange information or to use remote specialised geo-processing functions that were inaccessible before. Such possibilities create new interesting opportunities in many areas of our everyday lives.

To benefit from this modern technology, during the last decade a great deal of effort has been directed towards the design and establishment of geo-information infrastructures. So far, the main purpose of developing these infrastructures has been, to make geographic data broadly available, accessible and shareable for a multiplicity of users from different application domains.

The notion of developing distributed applications complementary to these infrastructures, which was initially neglected, is starting to draw attention. This is an important development specially because experience has shown that such infrastructures can only become useful and profitable, when they are deployed in the context of a variety of end-user application services.

New initiatives of distributed geo-processing have consequently appeared, namely distributed GIS (Geographic Information Systems), internet GIS, and more recently, there have been some developments in the area of web services. Unfortunately, a lot of attention has been given to implement distributed geo-components, partially disregarding the development and provision of guidelines to handle the reuse and combination of available geo-components and the design of new geo-components.

There exist consequently a need focus on the use of techniques to conceptualise distributed geo-processing systems as an important step in their development process. This would lead to obtaining a conceptual description also called abstract specification of every system part and every system function (internal or external). Such descriptions could be developed according to, for example, different sets of concerns or points of interest.

Two important considerations have to be taken into account for the conceptual design of high quality distributed geo-processing systems: (1) the specification and development of composable and shareable artefacts, viz., data, processes, operations, resources, value-added products; (2) the construction of geo-information services by assembling these prefabricated and configurable artefacts, enhancing the degree of tailorability provided by these systems.

Both (1) and (2) can be achieved by following an appropriate design methodology.

This thesis aims at the development of a methodology to support the design of distributed geo-information services. The methodology features a systematic approach to master complex designs, and incorporates proper concepts that enable the effective structuring of such designs. A system architecture upon which to deployed the designs is also proposed. This architecture is used to help deriving the necessary abstraction levels and identifying the various milestones.

The methodology is organised around two perspectives, the internal and external perspective. These perspectives are used to focus the design activities on specifics aspects of the system. A set of general purpose design concepts is provided to capture relevant information about the system according to the concerns associated with these perspectives.

The methodology proposes the use of a repository service to organise the creation, updating, validation, accessing and sharing of service models and service instances. The repository defines necessary and sufficient conditions for services to interact, while leaving maximum freedom to implementors to realise these services.

Acknowledgements

There are a number of people who has, in one way or another, contributed to make this work possible. I would like to thank some of them here.

First, I want to express my sincere gratitude to my promoter Prof. dr. Chris Vissers for his contribution, support, guidance and encouragement throughout the duration of this work. I would also like to thank Dr. Luís Ferreira Pires and Dr. Marten van Sinderen for their ever-lasting support and for providing me with outstanding professional guidance, feedback and advice even beyond the topics of this thesis. Their contribution was indispensable for the completion of this work as was their stimulating personal friendship.

A special word of thanks is due to Dr. Radwan who created the opportunity and laid the foundation for this work. He has been a continuous source of inspiration and has continuously provided me with the professional and personal support required to finalise this work. It was a privilege to have him as my supervisor and more importantly as an outstanding friend.

I am deeply indebted to the Geographic Institute Agustín Codazzi, IGAC, for offering me the chance to develop the ideas conveyed in this thesis. My gratitude goes to Santiago Borrero, former Director General of IGAC, for seeing the importance of my studies and for providing me with his endorsement and support. I want to express my sincerest gratitude to Amparo Figueroa and Ana Lucia Vallejo for their endless support and contribution to overcome the operational hurdles surrounding my studies.

I would also like to thank Chris Paresi, Dr. Liesbeth Kosters, Prof. dr. Martin Hale, and Loes Colenbrander for their support. I want to thank all my colleagues in the former GMI department, the members of educational affairs and all members of the PhD community at ITC.

I want to thank my direct circle of friends who provided me with their friendship, time and advice to escape from the long and lonely hours of work, and who became a second family in this far away land.

I extend my very special thanks to Dr. Rolf de By who lightened the spark back in 1997 at the very beginning of my research carrier, and who provided me with views, feedback and valuable professional and scientific advice. His support throughout the duration of this research was invaluable but I would especially like to thank him for being such a great friend.

“Quiero extender el mayor de los agradecimientos a mi querida madre, Socorro de Morales, por su cariño, sus oraciones y consejos, por su respaldo y comprensión y por estar allí siempre

que el camino se hizo duro y la carga mas pesada. Mama, he dedicado este trabajo a ti y a la memoria de mi padre"

Further, I can not thank my wife, Liliana, enough for her patience and support during all these years, but more importantly, I must thank her for her sacrifice in putting aside her own professional aspirations and committing fully to our family and to the cause of this endeavour. It is beyond any doubt that I would not have been able to complete this work without her encouragement, care, company and love. I owe you more than I will ever be able to repay. Finally, I want to thank our children Diego Nicolas and Maria Paula for their understanding, for condoning my absences, for providing me with their smiles and for being a continuous source of inspiration.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Anything I've ever done
that ultimately was worthwhile...
initially scared me to death.

Betty Bender

In this chapter we introduce the research accounted for in this thesis by explaining the motivation, introducing the objectives and illustrating the approach followed to achieve the objectives. The chapter opens in section 1.1 with the description of the motivation for the work; section 1.2 analyses trends in geo-information systems; section 1.3 introduces some terminology used throughout the thesis; section 1.4 discusses system development, development methods and the main aspects of development methodologies; section 1.5 defines the research objectives; section 1.6 outlines the approach; the chapter closes with the presentation of the overall structure of the thesis in section 1.7.

1.1 Motivation

We are living in an age where information plays a central role in our daily activities, driving and constraining every decision that we make. A good indication of the impact of information awareness can be seen, for example, at the financial markets, where interest rate values and share transactions change as some international facts become known.

We can also see information as an important decision factor in a more familiar situation, for instance when driving along the highway, where you make changes in your planned route as a traffic report becomes available. In this second example however,

one may require more information than just the traffic data. To make a satisfactory change of route chances are that you will need some additional information such as data about the road network, possibly some relief characteristics (the topography of the area you are in), proximity to built-up areas (cities, towns). This extra information is known as geographic information or geo-information .

Geo-information has always been used by individuals when they are having to find their way in unfamiliar territories. But beyond that, geo-information was almost exclusively used and valued by professionals, e.g., the military or the city planners, who exploit it to support some of their specific activities. This particularly exclusive use of geo-information by these so called traditional users was caused by the high cost and effort required to access these data and manipulate it properly.

During the last decade however, technological improvements have been facilitating the access to geo-information, and have been reducing the effort and skills required to use it effectively. As a consequence, the use of geo-information is expanding beyond the traditional users, to include new user communities. These new user communities now see how their activities can benefit from the proper use of geo-information, mainly because they have easier access to it and to the tools needed for its manipulation. Examples of these user communities include, among others, telecommunications, emergency services, transport companies and tourism.

As individuals start to have easier access to geo-information, a more sophisticated spatial awareness is being developed. This results in geo-information being required to support many daily activities. Nowadays, it is not uncommon to hear questions like, "which areas of the city are not covered by our retail offices?" or "how much overlap is there between retail offices A and B" or "what is the shortest route to supply all our retail offices." All these questions have location as a key ingredient. This growing dependence of people and organisations on geo-information has converted it into a precious resource.

Traditional users differ form other users in the sense that they know how geographic data is collected and represented, and they have a specific use for it. Consequently, they have well-defined requirements and furthermore, these requirements are to a large extent fulfilled by the conventional methods of geo-information production used by geo-information providers.

Nevertheless, as the use of information increases, these traditional users are identifying new application areas where geo-information can be exploited, hence, expanding their set of requirements making them more diverse and difficult to satisfy. Moreover, other users who are less aware of the nature of the data and require it for a variety of applications, turn to have a much wider and less specific set of requirements. This forces geo-information providers to deal with a large variety of information requirements to satisfy. It is a challenge for any provider to satisfy these requirements, especially if we take into account that the way information is perceived, expected and used depends very much on the current forms and shapes of markets, projects, and technology.

Most of the existing information providers rely on supportive systems (geo-information systems) that have been specifically designed to deal with a standard set of requirements, mainly those of traditional users. Systems that deal with requirements of this sort, are designed using system design methods that put the emphasis on static aspects of the system, such as data and structure. This has been traditionally the case with the design of geo-information systems. As a result these systems produce specific, static and pre-defined products. Unfortunately, these ever-lasting products do no longer fulfil the requirements of the users.

The requirements for these systems no longer remain static, but they keep on changing, mainly because users want to influence the products in many different ways, and because new technology offers a bunch of opportunities. Therefore, we have to extend the methodologies for geo-information system design such that these methodologies can support the development of systems that can cope with dynamically changing requirements.

This change can be achieved using a systematic methodology that pays attention not only to the structural aspects of the system but also to the system's behaviour. Analysing behaviour and possibly separating different behavioural patterns that might be reused, we can identify elements that could be assembled in multiple combinations, facilitating the generation of a larger set of services. Based on this idea we can design more flexible and adaptive architectures, where we can benefit from functionality reuse, and we can generate more tailored services to satisfy a wider group of users.

1.2 Geo-information systems

Geographic data fundamentally encompasses people's perception of real world phenomena such as, rivers, roads, cities, etc. To store information about geographic objects in an information system we use spatial data types. Spatial data types are a specialisation of traditional data types such as records or structured text. These data types represent geographic objects as geometric features with an associated location.

Geo-information systems are information systems that store and manage information about geographic objects. This information is mainly collected, maintained and provided to geo-information users who employ this information to solve spatial problems. Spatial problems are those which make implicit or explicit reference to locations or positions relative to the Earth.

Before technological advances like the World Wide Web emerged, geo-information was not very accessible for the general public, and it was stored at isolated information systems and in proprietary formats. With the advent of the World Wide Web, developments in data formats and data transmission followed, which have greatly facilitated access to and have increased the availability of geo-information. As a result, users do not have to rely on single data sources, but they can solve their spatial problems using multiple interconnected collections of geographic data.

The same technological advances have also opened the possibility to use remote geo-processing functions. Many specialised operations that were inaccessible before can now be accessed and exploited by users, who can incorporate them in their specific problem-solving approaches.

We see future geo-information systems as distributed systems from which specialised geo-information services can be generated by exploiting artefacts (data and functions) that are located in an infrastructure of interconnected service nodes. These artefacts are combined to define large geo-processing tasks that provide a more diverse functionality than that of the artefacts in isolation. From all of these considerations we can derive some important issues to concentrate on when proposing new directions in geo-information system design: composition, distribution, coordination and reuse.

The development of systems with these characteristics can be rather complex, therefore we require the use of structured methods to analyse, model, design and redesign the system. With the help of these methods, knowledge about the artefacts of the system and their relevant characteristics can be captured. This makes it possible to specify complex interactions between these artefacts for the generation of system services.

We distinguish three different classes of artefacts: data elements, processing elements and connecting elements. Processing elements generate or transform data elements; data elements contain the information that is used or manipulated by the processing elements; connecting elements represent the properties and constraints that govern the interactions between elements.

1.3 Systems and distributed systems

Organisations rely on different systems to support their activities, and to assist them in obtaining the information they need. Here we introduce a precise notion of system. A *system* is an entity formed by a configuration of interacting parts, put together to serve a purpose. A part of a system can be considered a system in itself and generally a system as a whole can be a subsystem or a component of a larger system.

In some cases, the parts of a system or subsystems have to operate somewhat independently from each other, which allows these parts, for example, to be placed close to geographically separated users, and still perform together as a single unit. In such cases the system is called a distributed system.

A *distributed system* is defined as a system whose parts are physically or logically separated and operate in a partly autonomous way. Each part of a distributed system exhibits a behaviour that is partially independent from other parts. Examples of distributed systems are organisations, communication networks and the World Wide Web.

1.4 Design methodology

A design methodology provides a set of structured procedures and rules that guide a designer in the process of designing a system. During this design process a series of models is created that represent the system and parts thereof. These models describe relevant aspects of the system under consideration, allowing for a better understanding of the system. In these models designers express their ideas as to how to achieve predefined design goals. An effective design methodology should support designers in the production of accurate designs in a target application domain.

A design methodology is defined as a collection of design methods based on design concepts and supported by design tools. Design concepts are used to represent primitive elements and their characteristics in an application domain. These design concepts are expressed via a textual or graphical design notation.

To enable the composition of design concepts, an underlying syntax is required that defines them precisely and unambiguously. This syntax enforces the proper use of the design concepts when they are combined to create designs. Designs are expressed by means of models, which in turn are written using a design notation.

A design methodology defines an organised set of guidelines, design steps and structuring techniques that are necessary to produce models of elements in an application domain. Tools provide (semi)-automated support for the production and analysis of these models, facilitating their creation, validation and verification.

The application domain defines the environment in which a design methodology is

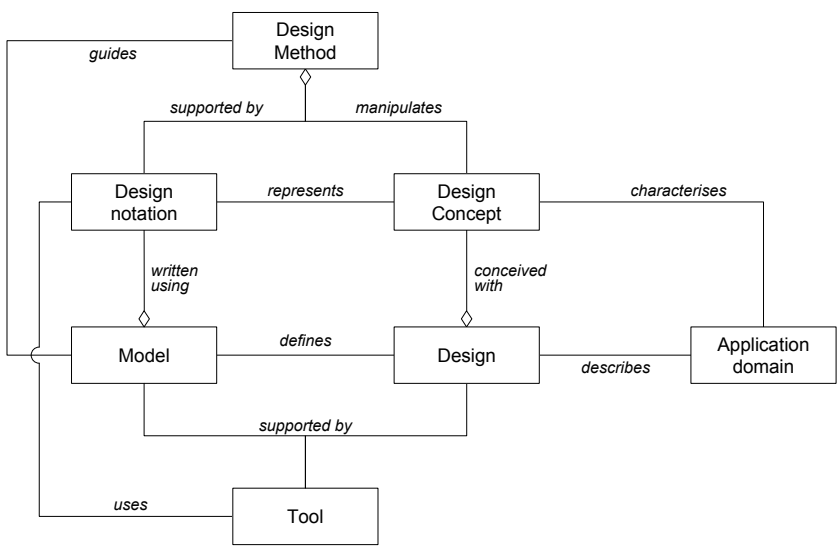


Figure 1.1: The elements of a design methodology

applied. It characterises the objects that are represented with the models created with the methodology. In our case these objects are distributed geo-information systems, parts of these systems, and the objects with which these systems interact. Figure 1.1 shows the relationships between the different elements comprising a design methodology.

Our group at the University of Twente has developed a methodology for the design of distributed systems that emphasises conceptual modelling of behaviour [FP94]. This methodology is supported by a consistent set of design concepts, and by a language that allows designers to represent instances of these concepts in designs. The design language is called Interaction Systems Design Language (ISDL) [QFPS02]. The language has been formalised [Qua98] in order to enforce precision and to allow comparison of designs. For example, the formalisation makes it possible to establish the mutual correctness of designs that are specified at different abstraction levels.

1.5 Objective

Modelling techniques allow the creation of models, of different aspects of a system, such as requirements, structure, or functionality. The generation of these models, prior to any implementation or improvement efforts, helps to achieve the proposed design goals.

In the case of a system that operates in a dynamic environment, where requirements are not stable, these models are of even greater importance. The reason is that such models can be the mechanism used to identify the system's parts and functions and manipulate them to obtain compliant architectures.

To facilitate the design of adaptable and conforming systems, system models should be generated in an integrated manner. A method is necessary to consider all aspects at various abstraction levels and therefore it helps to enforce the adaptability and compliance with changing requirements.

This research has focused on the development of a design methodology for geo-information systems. We do this by defining concepts and structuring techniques to support the specification and development process of geo-information systems. Specific objectives of our research have been:

- To identify suitable architectural concepts that allow developers to model and specify geo-information systems;
- To propose a suitable system architecture for the provision of geo-information services;
- To define a methodology that supports abstraction, modularity and other structuring mechanisms and that uses architectural concepts, for capturing knowl-

edge about the fundamental elements of geo-information systems. Such knowledge is represented in models that express structure and behaviour of the system;

- To develop a framework for the use of system elements as building blocks in multiple model-driven compositions for the construction of complex services.

1.6 Approach

The approach we adopted to achieve our objectives is as follows:

- Investigate the state-of-the-art in geo-information systems and geo-information sharing. This investigation is used as the basis to introduce a geo-information services architecture;
- Study the different ways of formalising systems, focusing especially on how to specify system behaviour and interaction systems;
- Introduce a methodology based on a sound design model that supports the development of geo-information systems. The methodology aids at managing the complexity of this system development process. This methodology helps to structure the basic elements of the system and provides guidelines to combine these elements systematically to form more complex element specifications;
- Propose a set of design concepts that can be used to create abstract models of system services. Services are specified by combining sets of predefined elements. These elements could be either elementary definitions or complex definitions, which were defined to provide an intermediate service. These models also serve to evaluate the functionality of a system, and as a template for implementation;
- Evaluate the methodology through the use of a comprehensive case study.

1.7 Structure of the thesis

The remaining chapters of this thesis are organised as follows:

Chapter 2 presents an overview of the state-of-the-art in geo-information systems with specific emphasis on geo-information infrastructures. The chapter also motivates and introduces the concept of geo-information service infrastructure (GSI) and a supporting architecture.

Chapter 3 discusses the concepts that underline system design. The chapter motivates the need for a process to guide the activities involved in developing systems. It provides an overview of techniques used to manage the complexity of the development

process. The chapter also identifies and analyses the various strategies available to steer the development process.

Chapter 4 introduces the Geo-Services Design Methodology (GSDM), which has been tailored to the development of geo-information systems.

Chapter 5 introduces the design concepts, and their combination rules, necessary to represent architectures of distributed systems.

Chapter 6 explains how to define GSI-services according to the external perspective. This perspective looks at the system from the point of view of its surrounding environment. The chapter also introduces a metamodel to facilitate the seamless interchange of service definitions.

Chapter 7 describes how to define GSI-services according to the internal perspective. The chapter explains the architectural style defined to guide and constrain the definition of components. The chapter shows how to use these design concepts to represent elements of a geo-information system. The chapter also includes the proposed strategies to assemble elements into chains to define complex services.

Chapter 8 provides a comprehensive example on the design of a Land Information Service that illustrates the main features of the methodology.

Chapter 9 summarises the research achievements, presents the conclusions, and provides some directions for further research and development.

Chapter 2

Geo-information concepts

To invent something,
you need a bit of imagination and a pile of junk.

Thomas A. Edison

This chapter presents an overview of state-of-the-art in geo-information systems and motivates and introduces the concept of geo-information service infrastructure. We start by introducing the concept of geo-information infrastructure as a mechanism to discover and access geographic data, with its main components, and then we examine each of those components in detail.

The chapter is organised as follows: section 2.1 deals with the issues of geo-information infrastructure; section 2.2 outlines the components of a GDI; section 2.3 explains geographic information in detail; section 2.4 concentrates on how we represent and structure geographic data; section 2.5 describes how to organise geographic data such that it can be used in a variety of applications; section 2.6 discusses aspects of documenting geographic data; and finally, section 2.7 wraps up the chapter by introducing the concept of the geo-information services infrastructure.

2.1 The geo-information infrastructure concept

Geographic information has always been a part of most cultures, because spatial thinking is an essential factor in people's relationship with the surrounding physical and cultural environment. This information is required in a wide variety of forms and contexts because linking location to information is a process that applies to many aspects of business and community decision-making. Individuals make temporary maps to remind themselves or show others how to find their way in an unfamiliar territory (way finding). The use of maps has increased as we develop sophisticated spatial

awareness and spatial communication abilities that came to support other activities besides the physical way finding. These other activities include urban planning, hazard management, intelligence gathering and cartography. The efficient creation, storage, processing, presentation and dissemination of geographic information is *'the'* big challenge these days. Having the ability to use this information efficiently is a strategic resource for now and into the future.

In the last decade, more and more volume of geographic data has been created to support geography-based analysis within multiple disciplines such as cartography, and cadastre. Due to advances in technology and to increasing users' awareness of the usefulness of geographic data, society's reliance on such data has grown substantially. This fact, together with the extended availability of methods and tools for the collection, processing, analysis and presentation of geographic data has led to a change in the way geo-information is produced, used and shared. The conventional approach of closed production systems with proprietary structure and purpose is changing into more open systems that allow data to be widely available for users with purposes possibly different from those for which the data was originally produced.

The opportunities for lower costs and shorter production times that were foreseen from the possibility of using existing data from different sources motivated users to share data. To satisfy this increasing need for data to be shared among multiple user communities and disciplines, a data sharing scheme has been put forward by some of the experts in the field. The idea was to establish a mechanism by which multiple collections of geographic data could be made available for everyone's use. This mechanism is known as the geo-information infrastructure or Geospatial Data Infrastructure (GDI).

A GDI is defined as the relevant base collection of technologies, policies and institutional arrangements that facilitate the availability of and access to geospatial data. The GDI provides the basis for spatial data discovery, evaluation, and application for users and providers within all levels of government, the commercial sector, the non-profit sector, academia and by citizens in general [Dou01].

The benefits of this mechanism are, a.o., reduced data collection times and cost, fast and wider access to data, interoperability between geo-applications and generation of data for multiple uses. Internet technology acted as a facilitator for the evolution of this concept into a tangible reality. Existing geospatial data infrastructures include:

- The Spatial Information Council – ANZLIC (Australia & New Zealand)[ANZ02];
- The Colombian Spatial Data Infrastructure – ICDE [ICD02];
- The Dutch Council for Geographic Information – RAVI [RAV02];
- The National System for Geographic Information – SNIG (Portugal) [SNI02];
- The National Geospatial Data Framework – NGDF (United Kingdom) [NGD02];
- The National Spatial Data Infrastructure – NSDI (USA) [FGD03];
- Common and Open (Geo-)Spatial Data Infrastructure – GDI-NRW (North Rhine Westphalia, Germany) [GDI02].

As the definition of GDI implies, such a data sharing scheme can only be realised if data producers at local, regional and/or national levels (depending on the scope of the GDI) agree on a set of legal, economical and technical principles that enable the discovery of and access to their geographic data.

2.2 GDI components

From the technical point of view, a GDI is a facility that acts as intermediary between producers and users of geographic data to facilitate data sharing. To make data sharing possible, data has to be made known, analysable and accessible. In order to realise these goals, various issues have to be addressed.

Existing data has to be organised to form data collections. A *data collection* is a set of structured or semi-structured geographic data collected by a data producer and stored in a proprietary format.

Data stored in data collections has to be described. Data producers with interest in sharing their data should create precise descriptions of their data collections. These descriptions should allow potential users to analyse and evaluate the collections' contents and determine their fitness for use. Such descriptions should also include information concerning the means to access and retrieve the data stored in the collection.

According to the principles behind the GDI initiatives, data descriptions as explained above are created in the form of metadata. *Metadata* is a formalised set of properties that describe with significant amount of detail the characteristics of the contents of a data collection.

When creating metadata, one follows a set of rules that delineate how to describe a collection of data (in our case geographic data), in terms of its contents, quality, con-

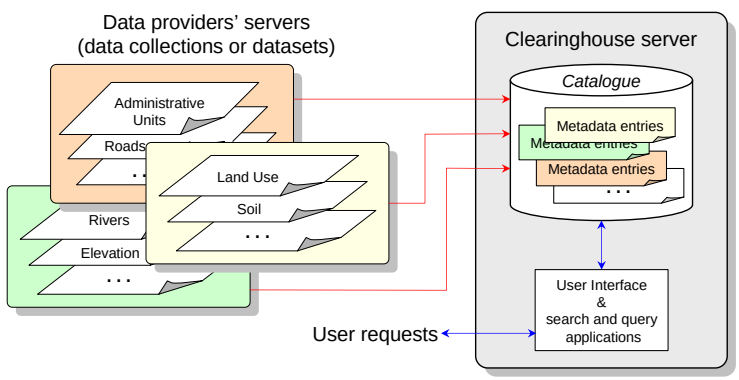


Figure 2.1: GDI components

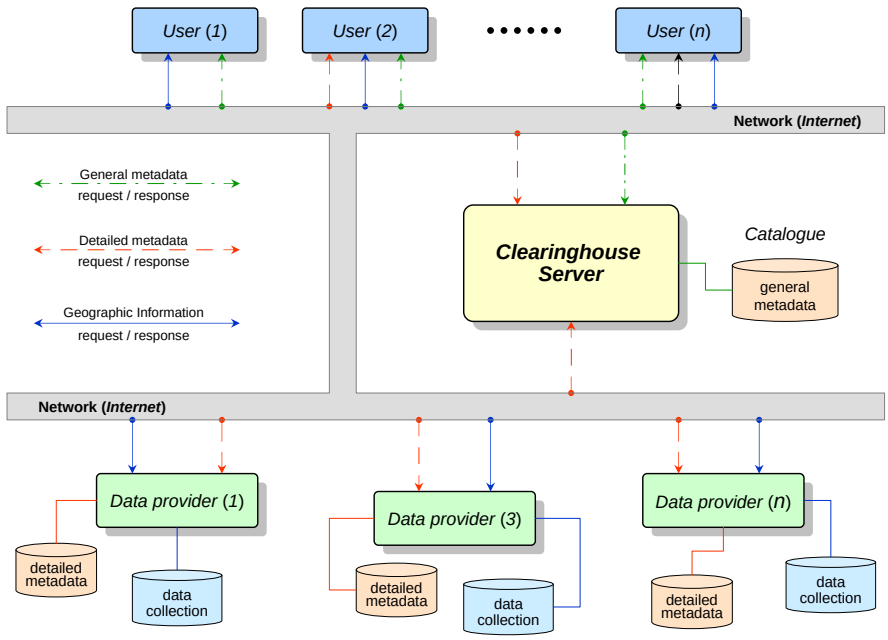


Figure 2.2: GDI typical architecture

dition, geographical extent, currentness and other factors of relevance to the potential users.

Different data providers wishing to make their data available through a GDI should follow the same set of rules to create their metadata. There exist various standards that provide common terminology and rules to describe geographic data. Some of these standards are described in section 2.6.2.

Within a GDI, data descriptions are organised and arranged as metadata entries (see Figure 2.1). A *metadata entry* contains information that describes a data collection. The arrangement of metadata entries results in a central catalogue that allows one to search and find data collections. A user interface allows users to interact with the catalogue to ask for and retrieve metadata entries. Figure 2.1 shows the relationships between the different elements mentioned above.

From the architectural point of view, data collections are located in distributed data servers and the catalogue is situated in an independent server that is commonly known as the clearinghouse server. Users can connect to the clearinghouse server to formulate their queries using, for example, a web browser.

Figure 2.2 shows one possible configuration of a GDI. The users post queries to the clearinghouse server to find out about availability of data. The queries are answered based on the contents of the catalogue, which in this case contains general (general)

metadata.

If a user finds a particular data collection useful for her/his application, s/he can request more specific descriptive details on that particular data collection. A search is then executed at the data provider site where more detailed metadata is stored to provide the user with the requested additional information. Figure 2.2 shows these two different query types; a dash-dot-line is used to represent a query to the catalogue (for general or global metadata), while a dashed-line represents queries for detailed metadata. The solid line represents requests for the data itself.

As a result of a query for metadata, a user obtains the following information: the name of the provider(s) that supply the required data; the web address(es) of the server(s) through which the data can be accessed; and, the conditions (e.g., price) and instructions on how the data can be retrieved.

2.3 Geographic data

Geographic data is the most important constituent of a GDI. *Geographic data* fundamentally encompasses people's perception of real world phenomena. These phenomena can be either natural such as trees, rivers, continents, rainfall and temperature, or man-made such as towns, buildings and roads. We make observations of these geographic elements to obtain relevant information about them. The information collected on these elements lets us determine, a.o., *what* they are, *where* they are and *what shape* they have.

Geographic data is used to perform multiple analysis mainly to support decision-making (e.g., where to locate a new railroad or how to get from A to B). Therefore it has to be organised and stored in a structured way. Normally, a collection of this data is arranged in descriptions composed of two parts: a *descriptive part*, consisting of a type or a name and all other specific attributes (the what), and a *positional part*, consisting of a geographic reference and certain shape determined by its boundaries (the where, the what shape).

Based on the information that we collect about geographic elements we see that these elements manifest themselves in two different ways: as clearly identifiable elements with clear-cut boundary, or as vaguely distinctive elements with indistinguishable boundary. In order to represent these two kinds of elements in a proper way they are classified in two groups: objects and fields, respectively.

Geographic objects are phenomena with well-defined spatial limits. This implies clear shape and size. Examples include buildings, railways, roads and water bodies. *Geographic fields*, in contrast, are those elements for which no distinct limit or boundary can be drawn. Examples are soil type, elevation, and temperature.

Objects and fields also differentiate from each other in the way their attribute values

Table 2.1: Selection of framework data for NGDI (source: [Ons02])

Country	Geographic Data Types																			
	1	***2***	***3***	***4***	***5***	***6***	***7***	8	9	10	11	***12***	13	14	15	16	17	18	19	20	21
Australia	●	●	●	●	●	●	●		○		○	●									
Canada	●	●	●	●	●	●	●	○		○	○	●				○					
Colombia	●	●		●	●	●	●					●	○								
Cyprus	●	●		●	●	●	●			○											
Finland	●	●			●									○	○						
France		●		●	●											○					
Germany	●	●	●	●	●																
Greece	●	●	●	●	●			○			○	●					○				
Hungary	●	●	●	●	●							●	○		○			○			
India	●																			○	
Indonesia	●	●		●	●				○			●									
Japan	●	●	●	●	●	●	●		○												
Kiribati		●			●																
Malaysia	●	●			●																
Mexico	●	●	●	●		●	●		○			●	○					○			
The Netherlands		●			●		●	○		○		●	○						○		
New Zealand	●	●			●		●				○			○							
Northern Ireland	●	●		●	●	●	●			○		●						○			
Russian Federation	●	●	●	●	●	●	●		○												
South Africa		●	●	●	●	●	●			○											
Sweden	●	●	●		●	●				○		●		○		○				○	○
United Kingdom	●	●	●	●	●	●	●	○		○	○	●				○					
United States	●	●	●	●	●	●	●		○												
Total (23)	***19***	***22***	***12***	***16***	***21***	***12***	***13***	4	6	7	5	***11***	4	3	2	4	1	3	1	2	1

●	selected for framework data of NGDI
○	not selected

Geographic Data Types

1.	***geodetic***	11.	place names
2.	***land surface elevation/topographic***	***12.***	***land use/land cover/vegetation***
3.	***digital imagery***	13.	geology
4.	***government boundaries/ administrative boundaries***	14.	real state price register/land valuation
		15.	land title register
5.	***cadastral/land ownership***	16.	postal address
6.	***transportation/roads***	17.	wetlands
7.	***hydrography/rivers and lakes planimetric***	18.	soils
		19.	register of private companies
8.	ocean coastlines	20.	gravity network
9.	bathymetry	21.	zoning and registration
10.	physical features/build		

are handled. Geographic objects are qualified using discrete attributes. For instance, a building has a number of stories, 5 or 6, a specific use, residential or industrial. Geographic fields, however, have attributes with values that vary continuously over space. This means that a different attribute value can be determined for different positions within the area represented by a field. Temperature, for example, varies in time and continuously over geographic areas.

We usually arrange geographic objects in groups. Such grouping depends very much on the views of particular user communities. Typical examples of these collections

are, the railways in a transportation system, the railway stations in that system, and the parcels in a cadastral system.

Geographic objects can also be organised at higher aggregation levels like, plots that form a neighbourhood, neighbourhoods that form a municipality. These aggregations can be useful for certain types of analysis, like capacity or connectivity computations. For example, if one needs to know how much water flows into a lake, rivers, streams and lakes will have to be studied together as a hydrographic system, to enable a query that can provide the required (capacity) answer.

Geographic data is collected to support a wide range of applications. Within the context of a GDI, however, the question arises whether data collected for one application can be effectively used for another. This consideration leads us to distinguish between three categories of geographic data: framework data, foundation data and application-specific data.

Framework data is collected with a broad audience in mind and can be used by multiple user communities with different expertise in multiple domains of interest. *Foundation data* is collected with the aim of satisfying the needs of a single user community with a particular expertise or domain of interest. *Application-specific* data is collected to serve no other purpose but an individual application with a highly specialised and narrow scope.

Table 2.1 [Ons02] shows the result of a survey in multiple countries to determine what data should be considered as framework data in the context of a National Geospatial Data Infrastructure (NGDI). Different countries selected from a list of geographic data types the most relevant ones (in terms of data sharing) according to their individual needs. These are shown as empty circles in Table 2.1. From this list 8 types were selected as being framework data, which are shown as filled circles in Table 2.1.

2.4 Representations of geographic data

This section introduces concepts required to create descriptions of geographic data such that this data can be organised and stored, e.g., in databases.

Digital representations of geometric data can be created using a variety of alternatives. This could be seen as a benefit because we can represent our data in the form that suits our specific application the best. However, if everyone chooses his own representation of data, the possibility of sharing data among multiple application domains diminishes considerably. This is because users and their intended applications are very diverse and, data collected and structured in an application-specific way becomes hard to re-use (see section 2.3).

The method selected to represent and store geographic data has huge implications on its possible uses. Therefore, it is important to understand the notions behind

geographic data representation. We use this notion in section 6.4 when we explain data definitions. The two most used computer representations of geographic data are *vector-based* and *raster-based* representations [By01, Wor95]. In the sequel, we describe in detail these representation methods, and we also discuss their suitability to represent geographic objects and geographic fields.

2.4.1 Vector-based representations

The representation of a geographic element consist of two parts: a descriptive part and a positional part (see section 2.3). In vector-based representations, an attempt is made to associate zero-, one- or two-dimensional features to the positional-part of geographic elements. This type of representation is also referred to as entity-based, feature-based or object-based representation. This representation method is the most suitable to handle geographic objects.

Zero-dimensional features or *points* are defined as a single coordinate pair (x, y) or coordinate triple (x, y, z), and are used to represent geographic elements with an area that is too small or irrelevant with respect to the scope of the spatial application. The values of the coordinates define the location of the element. Transmission towers, water valves, cities, airports, museums, churches are some of the elements that are often represented as points.

One-dimensional features or *lines* are defined by two delimiting end nodes and zero or more internal nodes or vertices that form segments or edges. Nodes are defined like points as discussed before, but they do not serve any purpose other than to help defining the shape of the line.

Linear features are used to represent geographic elements such as roads, rivers, pipelines, power lines, railroads. Multiple connected lines are often used for representing networks, especially when there is the need to study, for example, connectivity in a road network, or, capacity when monitoring a river system. Figure 2.3 shows a line representation with four segments connected by three vertices and delimited by two end nodes.

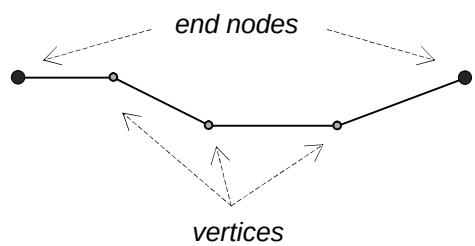


Figure 2.3: Linear feature representation

arc	from	to	left	right	vertices
a	2	5	A	B	···
b	1	2	A	D	···
c	1	4	D	C	···
d	5	6	E	B	···
e	3	1	A	C	···
f	5	3	A	E	···
g	4	2	D	B	···
h	6	4	C	B	···
i	3	6	C	E	···

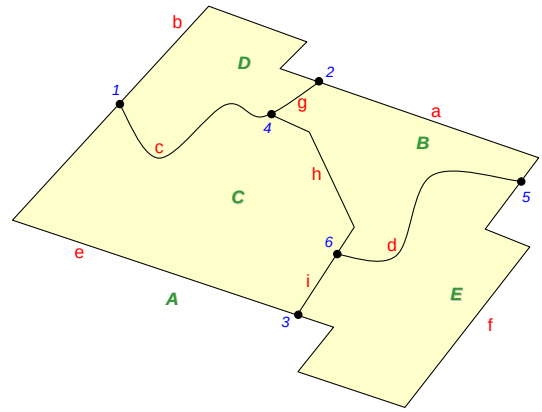


Figure 2.4: Region features representation

Two-dimensional features or *regions* are defined by a set of interconnected arcs (lines) that determine the boundary of the polygon that encloses a region. For every arc, we store its external nodes, its left and right polygons and the list of vertices. This representation form is called the *boundary model*.

Figure 2.4 shows a group of regions B, C, D, E. The list of arcs contains the initial and final nodes, in the columns left and right, that define the direction in which arcs have been represented, their left and right polygons, and the list of vertices (which have been omitted in Figure 2.4). Arc d, for example, starts in node 5 and ends in node 6, and it has polygon E to the left and polygon B to the right. Polygon A in the table denotes the outside polygon. The outside polygon represents the area surrounding the area of interest, or the external area.

The boundary model is often called the topological model since it captures some topological information, like, e.g., polygon neighbourhood. Region features are mostly used to depict geographic elements with large areas like lakes, parcels and administrative units.

Figure 2.5 shows some geographic elements represented using points, lines and regions, with a corresponding attribute value. The lake element in Figure 2.5, for example, is represented by a region feature and has the attribute value *4*. The house and the transmission tower are portrayed as points, while the road and railroad are depicted as lines.

The decision on which type of feature to use to represent an element depends on the application requirements. Here, the house is represented as a point, but if we need to make some analysis based on the area of the house for taxation purposes, then it is necessary to represent the house as a region. Shape and size of the elements can be represented more accurately with this type of representation.

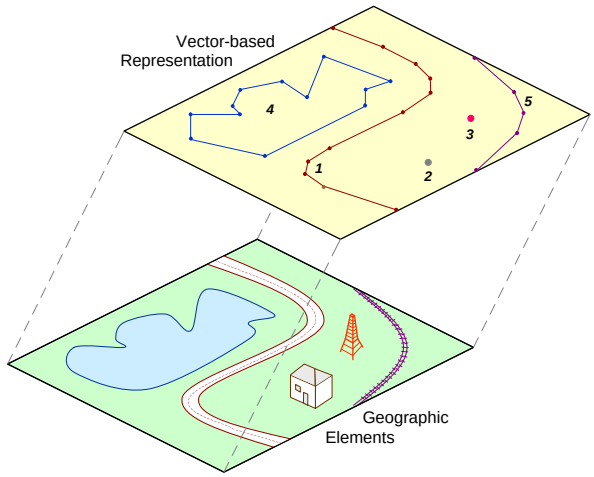


Figure 2.5: Vector-based representation of geographic data

2.4.2 Raster-based representations

The raster-based representation approach makes use of *tessellations* for the representation of geographic elements. A tessellation can be defined as a repeating pattern of interconnected shapes. With a tessellation, the whole study area is partitioned into an arrangement of pairwise, disjoint cells. Each cell is associated with attribute values or labels that indicate to which geographic element it belongs.

Tessellations can be regular or irregular. A regular tessellation is formed by cells that are congruent regular polygons. Regular means that the sides of the polygon are all the same length, and congruent means that also all the polygons are of the same size and shape, such as in Figure 2.6.

Irregular tessellations are partitions of the study space into disjoint cells, but in this case the cells may vary in shape and size. A well-known form of irregular tessellation is the *region quadtree*. It is also made up of square cells, but in this case adjacent cells with the same attribute value or label are merged to form a bigger cell.

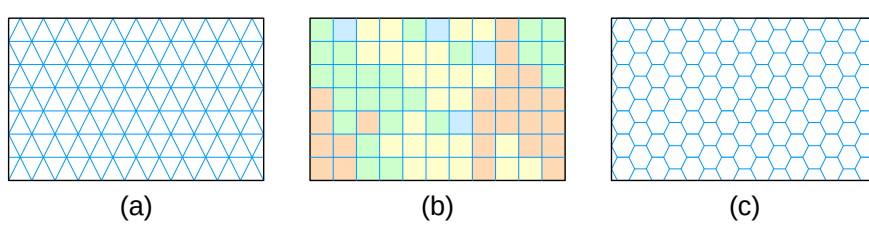


Figure 2.6: Regular tessellations: (a) triangular cells, (b) square cells, and (c) hexagonal cells.

Square cell tessellations, also known as *grids* or *rasters*, are mostly used to represent geographic fields (elevation, temperature, etc). Grids are suitable for representing area objects (polygons). In this case, each cell in the grid is labelled according to the polygon it belongs to. Grids are less suitable to represent geographic objects with point or line geometry.

Grids greatly facilitate tasks such as geo-referencing, because in a grid of $m \times n$ cells the location that corresponds to a particular cell can be determined rather easily, just by knowing the cell position (row and column coordinates), and the cell size.

The cell size of a tessellation is an important aspect to consider. Various factors may influence the choice of the cell size; the most relevant factors are the scale required for the intended application, the resolution of the elements of interest and the type of data to be obtained such as satellite imagery. A small cell size yields a representation close to the geometry of the real objects, but results in a large data set that may be difficult to manipulate.

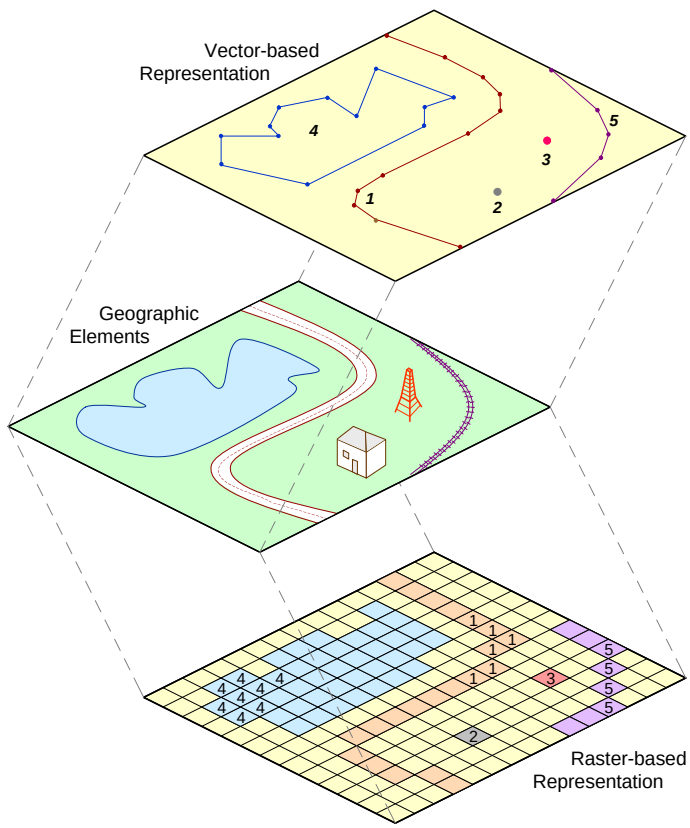


Figure 2.7: Representations of geographic data

It is always necessary to make a trade-off between the accuracy of the representation of the elements portrayed in the tessellation and the amount of memory space required to store and manipulate the data, specially when dealing with large areas. The processing time for most of the operations on tessellations increases more or less linearly with the amount of data.

Figure 2.7 shows how the geometry of a group of geographic elements is represented using both vector-based and raster-based representations. The bottom of the figure depicts how in the raster-based approach a geographic element is represented by a series of neighbouring cells that share the same attribute value. For instance, the cells in the grid that represent the *road* have been assigned the value *1*, the cells that represent the *lake* have been assigned the value *4*, etc. The figure also shows how the cell size affects the geometric representation that is associated with the various elements.

2.5 Organisation of geographic data

The most optimal way to organise large amounts of data, either geographic or of any other type, so that it can be accessed, managed, retrieved and updated is in a database. Databases are organised around data models. Data models represent people's perception of reality in an abstract way. Data models are used to describe the architecture of a database. By architecture we mean the data contents, relationships and constraints, the data structure and the physical storage or data format. Data models are commonly organised into different abstraction levels according to what aspect of the database architecture they describe.

2.5.1 Conceptual schema

The conceptual level describes what information is stored in the database. At this level we define precisely what are the data contents, the relationships among the data, and the constraints that should hold on this data. This description of the database is called the *conceptual database schema*.

In a conceptual schema, we represent a set of real world objects (employees, rivers, buildings) or concepts (projects, temperature, rainfall) of interest. We identify properties of interest that further describe the objects and concepts, such as the buildings's name and owner. We also describe relationships among objects, for example, an is-owned-by relationship between a building and an owner.

Conceptual schemas can be expressed using different techniques, the most common ones being Entity-Relationship (ER) diagrams and class diagrams (UML). In ER diagrams we use concepts such as entity types, attributes and relationships. *Entity types* represent objects or concepts, *attributes* represent descriptive properties of the entities, and *relationships* represent interactions among entities.

In class diagrams [OMG01d] we use concepts such as objects, classes, attributes, operations and associations. to create conceptual data models. An *object* represents an identifiable abstract or concrete thing. A *class* describes a set of objects that share the same attributes, operations, methods and relationships. A class is used to group objects in such a way that their similarities can be emphasised and their differences ignored. *Attributes* are named properties of objects that contain values that qualify the individual objects. *Associations* represent static relationships that exist among classes.

2.5.2 Building geographic databases

Figure 2.8 shows a selection of elements from the real world, and a conceptual schema of those elements constructed using a class diagram, which captures the types of objects of interest and some of their relationships from the real world. Figure 2.8 also shows that a logical data model can be built from the conceptual data model using the relational model technique. The logical model shows a group of tables, where the rows in the tables represent objects that belong to a certain class. The lines connecting the tables represent relationships between the objects, such as, who owns which building or which road gives access to which building.

The process of designing a database to store geographic data is fundamentally the same as that of any other database. However, one additional issue that we consider in geospatial databases is the definition of the spatial representation of the objects or concepts of interest together with their spatial and topological relationships [Zei99, RSV01]. This means that after we have created the conceptual schema and identified objects and relationships, we determine whether these objects are represented as

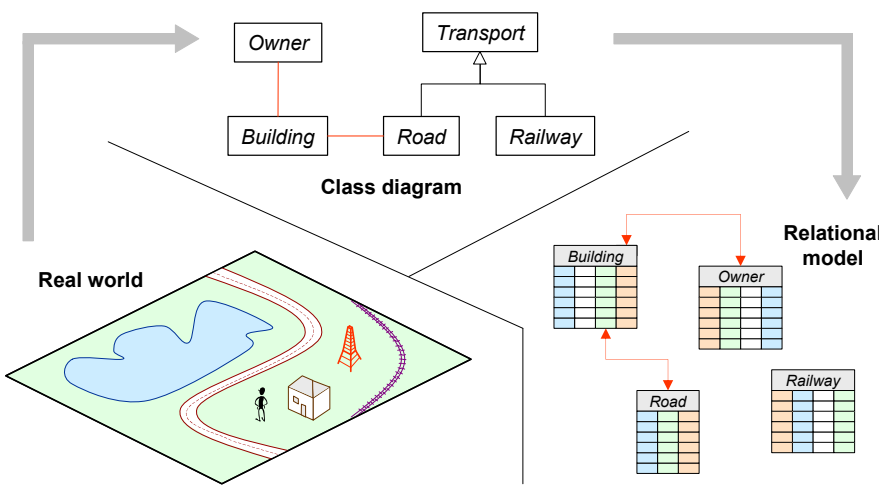


Figure 2.8: Conceptual and logical abstraction levels

points, lines, polygons or grids. This decision depends, among others, on the nature of the objects, the significance of their shape, the scale (spatial scale) of the application (see section 2.4). We also define topological associations such as, for example, groups of elements, e.g., bus stop, bus station, street and traffic light, which together form a transportation network.

One last issue in the design of a geographic database is the determination and assignment of the coordinate system that references spatia;;y the contents of the database. A coordinate system, is used to define locations on earth. Based on a coordinate system objects can be referenced by their coordinates with respect to an origin point. Locations of objects are identified by x,y coordinates on a grid, with the origin at the centre of the grid. One specifies its horizontal position (East–West) and the other its vertical position (North–South). The two values are called the x-coordinate and y-coordinate. Using this notation, the coordinates at an origin could be $x = 1000000$ and $y = 1000000$.

2.6 Describing geographic data

The value of geographic data is recognised by both government and society, but its effective use has remained inhibited by poor knowledge of the existence and whereabouts of data, poorly documented data collections, and data inconsistencies.

Once created, geographic data can be used by multiple users for different purposes, given that its existence and fitness-for-use are known. If we also consider the dynamic nature of this type of data, we conclude that describing the data is therefore an essential requirement. For a community organised around the concept of data sharing, data plus its documentation (metadata) is certainly more valuable than just undocumented data.

Metadata can be used by data providers to monitor and control the status as well as to advertise their data to potential users. Coordinated metadata development and advertisement could avoid duplication of effort and waste of resources, by ensuring that producers are aware of the existence of data collections.

Metadata enables users to locate all available geographic and associated data covering their particular area of interest, therefore improving the quality of their analysis and applications. Generating metadata certainly adds to the cost of data collection, but in the long run the value of the data or the revenues that can be obtained from the data increase if that data is properly documented and described by means of metadata.

The term metadata and the metadata itself have become widely used over the last years, but its underlying concepts have been in use since collections of information started being created and properly organised. Library catalogues represent an established sort of metadata that has served for decades as a management and resource discovery mechanism.

The concept of metadata is now common among people who deal with location related issues. A map legend is one representation of metadata, containing information about the publisher of the map, the publication date, the type of map, a description of the map, spatial references and the map's scale and accuracy, among other things. Metadata can be defined as the set of descriptive information applied to a data collection that enables its discovery, access and use.

2.6.1 Levels of metadata

Based on how metadata is used, we organise it in three different levels:

- *Discovery metadata* allows users to find data collections that contain the sort of geographic data they are interested in. Producers publish or make available this type of metadata to advertise the contents of their data collections;
- *Exploration metadata* allows users to perform a deeper study of discovered data collections, to determine if or which of the data is useful for their purposes. This type of metadata aims at ensuring that data is used correctly and wisely;
- *Exploitation metadata* allows users to determine how to obtain and use the data that they suspect can satisfy their needs. This type of metadata aims at ensuring that existing data can be accessed.

We are not implying that these levels of metadata are unique, but rather we find them sufficient as description levels of data collections. The boundaries between these levels are not sharp or definite and the levels are somehow complementary or overlapping. Data producers should be aware of these issues and they should look at their overall needs and requirements, and then determine and specify how their metadata should be structured, or, in other words, what data to collect as metadata.

Discovery metadata is the minimum amount of information that needs to be provided to inform anyone interested in the nature and content of a data collection. This means that discovery metadata should address the issues of "who has what data and from where." *Who* defines the creator and supplier of the data, and possibly intended audience; *what* defines a name or title, resolution and description of the data; *where* defines the geographical area being covered, based on coordinates, geographical names or administrative areas. Discovery metadata usually, but not exclusively, enables others to discover collections of data that have similar contents and that cover (describe) either the same, or similar, or partially overlapping geographic areas.

Exploration metadata provides sufficient information to enable users to determine whether some existing data is useful for their application. This means that exploration metadata should address the issues of "why, when and how was data collected." *Why* defines the reasons for data collection and its intended uses; *when* defines the date that the data represents, when the data set was created and the update sequence, if

any; *how* defines what instruments, procedures and sources were used to generate the data, and how the data is structured.

Exploration metadata includes the details required to allow the prospective end-users to know whether the data meets the requirements for their application. Exploration metadata usually facilitates to establish the quality and usability of data collections and whether they are up-to-date.

Exploitation metadata provides information to enable users to access, transfer, load, interpret and use the data in their end applications. This type of metadata also includes details on contact person, the price of the data and copyright.

2.6.2 Metadata standards

In order to make metadata readable, understandable and usable by users from different disciplines, standards are necessary to provide a common terminology and to define how to document geographic data. To the best of our knowledge there exist at the moment three main metadata standards of broad international scope and usage:

- CSDGM, the Content Standard for Digital Geospatial Metadata [FGD98], created in the USA by the Federal Geographic Data Committee (FGDC) in 1994.
- ENV, the European Set of Standards for Metadata Generation (Euro-Norme Voluntaire), created by the Comité Européen de Normalisation (CEN) in 1998.
- ISO-FDIS 19115, the International Standard for Geographic Information - Metadata [ISO01c], developed by ISO through the Technical Committee 211 (ISO TC-211), which is responsible for the Geoinformation/Geomatics area.

Metadata also forms an important part of the OpenGIS Abstract Specification. The OpenGIS Consortium (OGC) is an international membership organisation engaged in a cooperative effort to create open computing specifications in the area of geoprocessing. As part of the draft 'OpenGIS Abstract Specification' there is a topic on recording metadata for spatial data, which is developed in cooperation with ISO-TC 211.

The ISO/FDIS 19115 - Draft International Standard [ISO01c] is now in its third version. It is the responsibility of 33 countries and 12 observer organisations to turn this draft standard into an International Standard. People expect that all the existing standards will converge through the ISO initiative. Indeed, most of the existing standards already have a great deal in common and a robust international discussion has ensured that the ISO standard has accommodated most of the various international requirements. The ISO standard has equally benefited from the experience of the various national bodies and their implementations of metadata standards.

2.7 Geo-information services

The role of the GDI is currently changing, from it being a simple data discovery and retrieval facility to become an integrated system suitable for the provision of customised information and services. We consider a service as the contribution of a system or part thereof to its surrounding environment. This contribution can be defined in terms of data, operations, processes, resources, value-added products or any combinations of them. For the sake of simplicity we use the term services to denote geo-information services.

Normally developers address the issue of designing complex services by stringing together groups of functions in an ad hoc manner. This approach may satisfy a particular need but doing this separately for different services hampers reusability. Moreover, lack of descriptions of the solutions obtained makes it hard to aggregate solutions to execute complex tasks.

Research is therefore focusing on the development of mechanisms to describe, combine and manage independent collections of services. Here, we introduce a concept that aims at facilitating the generation of sophisticated value-added services. We call it the Geo-information Service Infrastructure or GSI for short. The idea of the GSI is that elementary services can be described, accessed, combined and managed to deliver complex content. Within the GSI, a common method is used to describe elementary services and their interfaces, and then these services are made available for users to create service chains that perform complex geo-processing tasks.

2.7.1 GSI

A Geo-information Service Infrastructure (see Figure 2.9) is a system from which specialised geo-information products and services can be obtained by exploiting the artefacts (see section 4.5) of an infrastructure of interconnected nodes that include data repositories, data brokers, service providers, service brokers and clients. This service framework builds upon the layer of interoperability of information as defined by the OpenGIS implementation specifications [OGC99], therefore separating the actual implementation of services from their definitions and the perception of these services by the users.

Large geo-processing tasks can be constructed by combining or chaining sets of artefacts located along the distributed nodes. Such combinations of artefacts provide diverse functionality that satisfies particular sets of requirements. Every artefact has an economic value; these artefacts are assembled to perform operations within the infrastructure, resulting in a specialised artefact that has a value equal or larger than the combined value of the artefacts used. This architectural approach can be regarded as a "value-added system." By chaining artefacts one can provide a service. A service is defined as a behaviour of value to the user, which is accessible or instantiated through interaction points (see section 5.1.2). This behaviour is exhibited through an

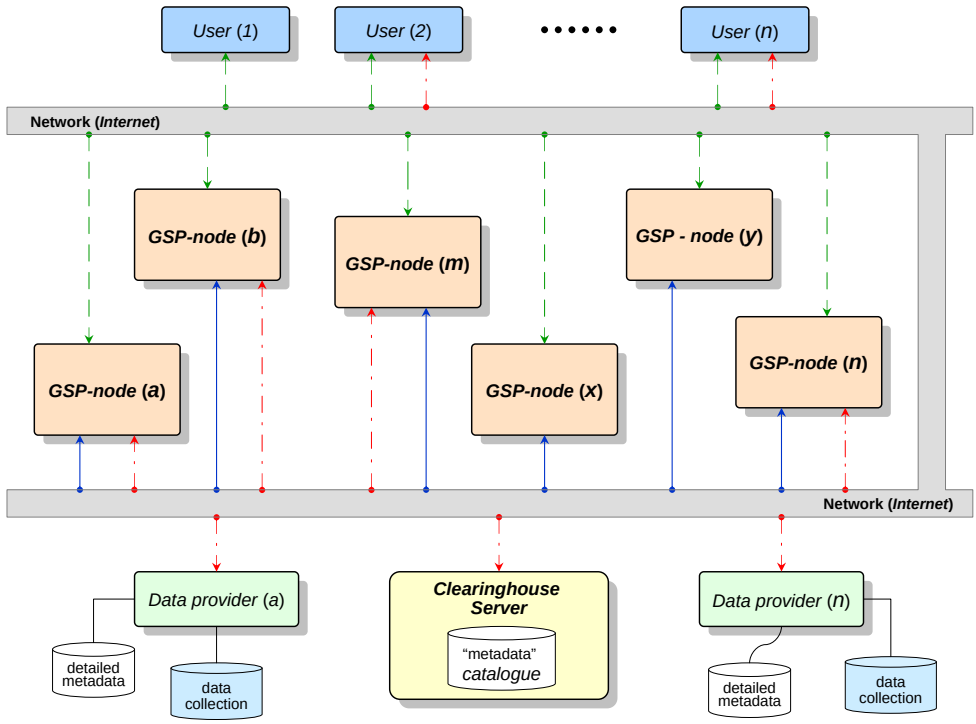


Figure 2.9: The GSI system concept

appropriate combination of elementary artefacts.

In order to bind multiple artefacts into a chain that accomplishes a large geo-processing task, a proper description of the participating artefacts is required. These descriptions focus on exposing the artefact's internal behaviour, its intended effect and its interaction points or points of composition. These descriptions, which are presented as instances of well-defined models, make it possible to interchange and reuse artefacts. We call these descriptions system metadata; they are stored and made accessible through a service repository.

The GSI system enables Geo-Service Providers (GSPs) to make use of functionality offered by others to supply a wide range of services and possibly to reach larger groups of users. Figure 2.9 illustrates the interactions that take place as GSP-nodes provide services to their users.

Users interact with the different GSP-nodes to request their specific services. Figure 2.9 shows these interactions as dashed-lines. GSP-nodes may make use of artefacts available in other GSP-nodes in order to realise a particular service. These interactions between GSP-nodes are shown in Figure 2.9 as solid lines running from node to node. All connections mentioned here are established through a network.

At the bottom of Figure 2.9 we can see that additional data collections located at non-GSP-nodes may still be accessed, if needed, either by users or service providers. This is achieved by making use of the conventional data discovery functionality, of the clearinghouse server. These interactions appear in Figure 2.9 as dashed-dot-lines.

2.7.2 GSP-node

Figure 2.10 shows the internal configuration of a GSP-node. The *service repository* contains the descriptions of available artefacts, either data definitions, process definitions, or previously assembled service chains.

The *geo-processing units* are responsible for the execution of the various functions of the node. These units use *geo-data* and *applications* during operation as specified in the definitions stored in the service repository. The *service design unit* is in charge of defining how the different services are realised. The underlying principles of this architecture are based on OGC technical baseline specifications for OpenGIS Services [OGC02].

The process of generating service chains within the GSI can be broken down into three major activities: defining and registering elementary services, assembling a service chain and delivering the results. Three different roles can be identified from these activities: service providers, service consumers and end users.

Service providers are responsible for describing and making their elementary services available for others to use. We denote the entities that provide these elementary services as components. Service providers make use of a framework (design methodology)

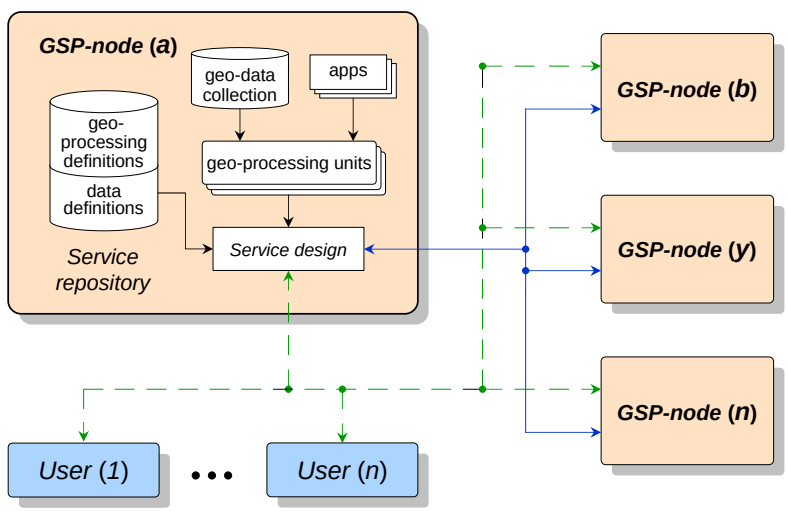


Figure 2.10: GSI-node internal structure

that enables the modelling of these components. These models act as descriptors that specify the function and the interaction point(s) of the components.

Service consumers use available components to design more complex services. Service consumers make use of the same framework used by the service providers to assemble individual components into chains. They define these chains by adding control components that govern the relations between the elementary components used in the chain.

These control components help ensuring that the constraints and conditions defined at the interaction points of the elementary components are satisfied. The resulting chain is described as a *service realisation*. The service can be realised by instantiating the behaviour portrayed in the specification. End users trigger the definition and execution of service specifications by posting requests to the system.

Chapter 3

System's representation and architecting

You can tell whether a man is clever by his answers. You can tell whether a man is wise by his questions.

Naguib Mahfouz

This chapter discusses the concepts that underline system design. The chapter motivates the need for a process to guide the activities involved in developing systems. It provides an overview of techniques used to manage the complexity of the development process. The chapter also identifies and analyses the various strategies available to steer the development process.

The chapter is organised as follows: section 3.1 motivates the need for a development process; section 3.2 introduces the concepts underlining system design and describes how techniques such as abstraction can be used to structure the development process; section 3.3 introduces and analyses the most popular development strategies.

3.1 System architecting

The primary purpose of system architecting is to help designers cope with the complexity of large systems and, more specifically, the complexity of those systems' development process. This is especially important when dealing with systems that support the core business of organisations, which is often the role of geo-information systems.

Complexity arises when systems can not be grasped in their entirety, mainly because they are formed by a vast number of parts that have numerous relationships among

them. In such cases, designers are forced to split the system into parts, such that each individual part can be studied and understood independently, and its interactions with other parts can be considered in isolation. Complexity does not decrease in this way but it does become manageable.

A *system* is defined as a collection of independent artefacts that interact and form a coherent entity that performs a function that is unachievable by any of the artefacts in isolation. *System architecting* is defined as the application of a systematic, well-structured approach to the development, operation and maintenance of a system.

A *system development process* is the set of activities followed by a team to analyse a problem and transform a set of customer requirements into a system (see Figure 3.1). This set of activities helps structuring the actions and thoughts of the development team by clearly stating what to do in terms of phases and outputs, when to do it, how to do it and why.

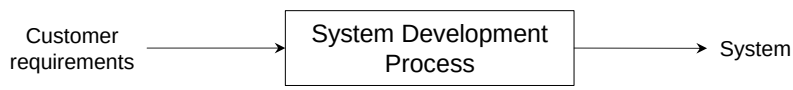


Figure 3.1: A system development process

The process of developing a system resembles very much that of construction of buildings or bridges. When constructing a building one does not start piling up bricks, but rather studies the needs of the client and then outlines a structure that complies to those needs. Information system construction goes much in the same way. First, the problem is analysed and the requirements identified and described in a precise way. Next, a design based on the requirements is made, and this design should conceptually solve the problem for the user. Next, the construction process starts and the actual system is realised. Finally the system is tested in a controlled environment to validate the fulfillment of the requirements. Generally, a system is said to be well-designed if it provides the expected (or predefined) service, within the limits of its budget and for a reasonable period of time.

These phases are not necessarily sequential. It is possible, for instance, that during the development process the requirements change or that errors are found and, therefore, backtracking to an earlier phase takes place. Moreover, the phases are not strictly separated. For example, it is quite possible to start the implementation of some parts of the system, which have been completely designed, while delaying the implementation of other parts, which have not been fully designed yet.

The goals of this separation into phases are, first, to provide adequate structuring of the various aspects addressed during development; and, second, to identify a number of design milestones that allow to control the progress of the development process, and the verification of the transition between subsequent phases.

3.2 Architectural principles

During the design process of a system we may decompose the system into smaller identifiable parts. There are, in principle, many ways to decompose a system into parts. Therefore, it is essential to establish, in addition to controlling complexity, the required design features of a decomposition such that a decomposition method can be applied correctly and satisfactorily.

The design characteristics we are most interested in are modularity, functionality, reuse and distribution. These characteristics are specially relevant for systems that may witness frequent change. A design process organised around these critical aspects is, in principle, capable of supporting the design of systems that are easy to maintain, flexible to adapt, while satisfying changing requirements.

3.2.1 Abstraction

The process of designing and developing a system like a vehicle, a database, a map, a software system or a telematic system involves the generation of descriptions of the various characteristics of the system under consideration. The object's descriptive characteristics initially exist only in the minds of the designers and need to be expressed, therefore, to express their ideas, designers make use of formalised notation to create a conceptual description of the system. In most situations, however, the amount of detail required to create a description that concretely and sufficiently describes a system simply overwhelms the minds of those working on it.

To deal with this problem, designers use a technique that enables them to reduce and organise the amount of detail required to create a particular description. This technique makes it possible to generate a series of descriptions of the same system, each focusing on the representation of a specific aspect () of the object and ignoring the other aspects. This technique is called *abstraction*. Abstraction can be defined as the process of selecting and representing particular aspects of a system or problem that are relevant at a particular phase of a development process while ignoring those other details that are also proper to the object but yet irrelevant during that phase of development. Abstraction is the primary technique for handling complexity.

Making the correct choice of abstraction facilitates the generation of clear representations of a system, which then enable people to make sense out of what might become a rather complex implementation. An accurate representation created using the correct abstraction choice is called a specification.

Figure 3.2 illustrates the concept of abstraction. In this example a description of a system is presented in the form of a map. The description represents an underground system. System's descriptions are always created for a specific purpose. This purpose should be used to select the choice of abstraction. The purpose of the underground description is to aid travellers in using the underground. For this reason in the underground description much of the physical detail has been removed and only the

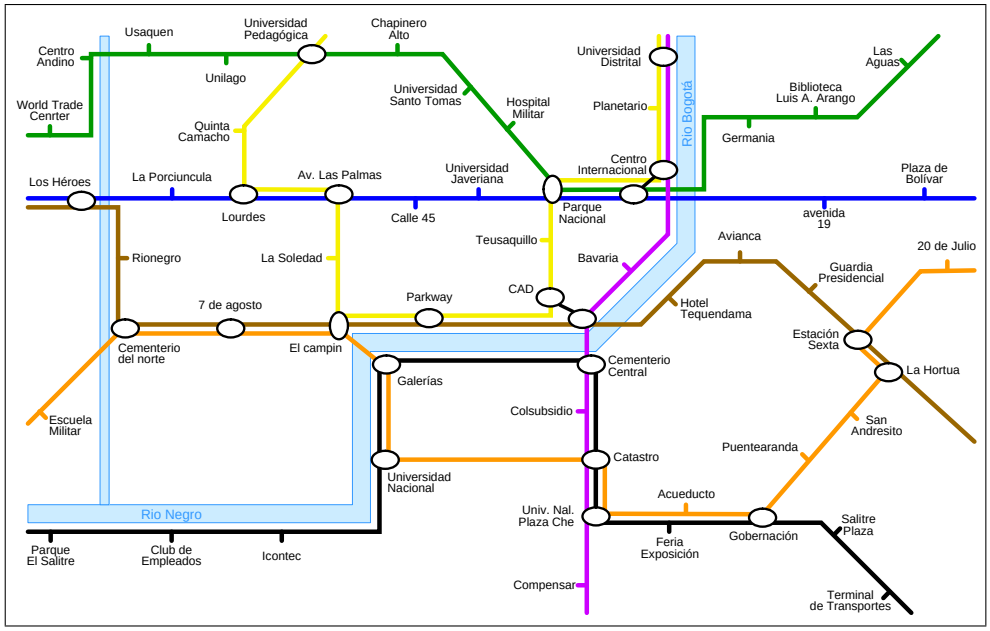


Figure 3.2: Abstract representation of an underground system

logical layout is presented. The logical layout (the relevant aspect in this case) is defined by the metro lines, the stations and the connections between stations.

When creating a description of a system, one selects those design concepts that best express the aspects of interest. In the underground description (Figure 3.2), abstract concepts such as colours and shapes are used to show metro lines and stations, and straight lines with 45° or 90° angles are used to show the connections between stations. We intentionally selected this example of the use of abstraction since it brings together the two central subjects of this research; geo-information and system design.

Abstraction is a familiar concept in the field of geo-information. Abstraction is the concept that underlines the techniques for representation of geographic information in computer systems. Moreover, abstraction has always been used in the field of geo-information as a tool to control what and how information is portrayed on maps.

3.2.2 A model of a system

Modelling is essential in system design. To be able to express and communicate the characteristics of a system that are relevant in a design project it is necessary to create system models (see Figure 3.3). A *model* is a conceptual (abstract) representation of a real-world system or part thereof. A model is the result of the designers mental image of a system. The model of a system considers some characteristics of the system, and ignores others. Abstraction as defined above can be used to capture the common

properties of a system, therefore, it is the main technique employed in the creation of system's models.

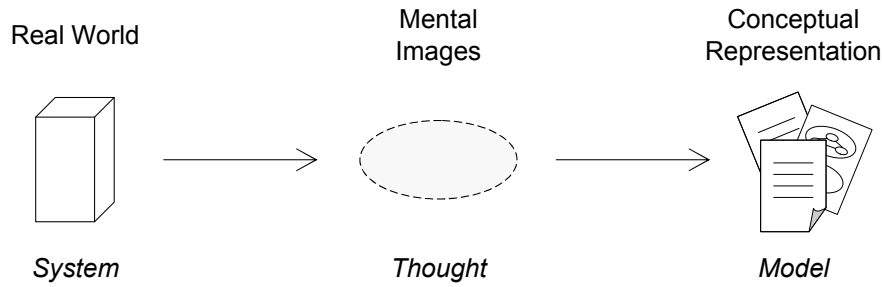


Figure 3.3: System and model

A model of a system is an abstraction of this system. A model can be created using different alternative representations. For example, a miniature of a house and its construction drawings are both models of the house. These representations are different because they have different purposes, e.g., the miniature is used to give potential buyers a good impression of how the house will look, while the construction drawings are used by the constructors to know what must be built.

A model can only be effectively used for communicating the characteristics of the system being modelled to the people interested in the system, if the model possesses the following properties [VFPQS99]:

- it is precise enough to avoid confusion on what it means (preciseness);
- it is simple and well-structured to be understood;
- it is clearly related to the system that it models, such that it appeals to the intuitive understanding of those who have to use it;
- it can be manipulated such that possible behaviour and properties of the system it models can be explored.

3.2.3 Abstraction levels

Although designers benefit from the use of abstraction when devising a new system, abstraction only helps to focus on a controllable amount of detail. Yet all the necessary details to fully express a design have to be considered and subsequently described to obtain the desired system.

To accommodate all the essential information for a complete design, multiple models have to be created. This can be done by using different levels of abstraction, each of them dealing with a different set of characteristics of the system. At each abstraction level models of the system are created, which consider the system at a predefined level of detail [BCK03].

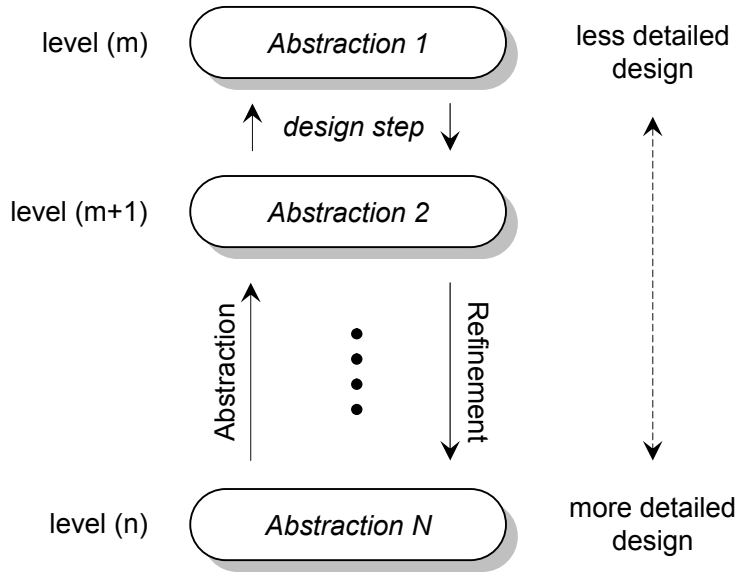


Figure 3.4: Abstraction, refinement and levels of abstraction

If a design is composed of multiple abstraction levels, then, given two abstraction levels (A and B), we say that A is at a higher level than B *if and only if*: 1) A expresses fewer properties than B; and, 2) all the properties of the system that are expressed in A are preserved in B.

The action of moving from one level of abstraction to the next level is called a *design step*. The process of moving from one level of abstraction to another to create a model that includes a larger amount of detail than the previous one is called refinement. *Refinement* is the reverse of abstraction, since it is used to incorporate the detail abstracted away in previous design steps, providing a more detailed description of the system. The relationship between abstraction, refinement and levels of abstraction is shown in Figure 3.4.

3.2.4 Views

In some cases, it is not practical to include all relevant details about a system for some abstraction level, for instance in a single monolithic design. For example, it may not be desirable to mix in a single system description information about what a system does (functionality), with information about how effectively the system operates (performance).

In such cases, one can use the concept of views. A *view* is a representation of a system relative to a set of concerns or points of interest. Instead of expressing everything about a system in a single model, a view addresses a subset of the concerns on the

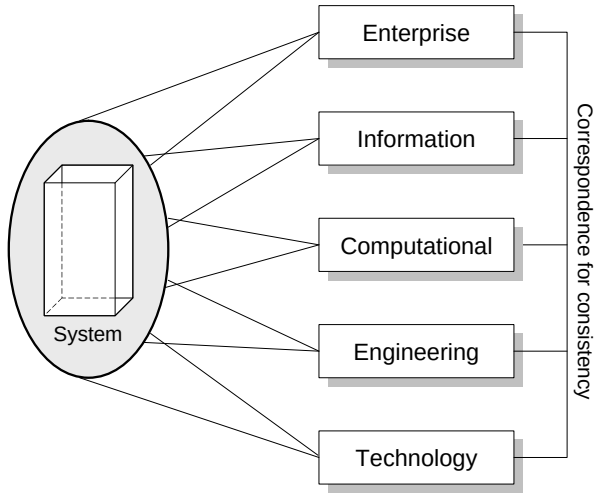


Figure 3.5: RM-ODP viewpoints on a system

whole system. These subsets of concern may be defined according to some development objectives, examples include, performance, cost, maintainability and reliability.

A set of concerns or aspects of interest is called a *viewpoint*. Viewpoints are not layered, but rather different perspectives of the same system. A view is a representation of a system according to a viewpoint. Each view is considered a complete representation of the system with respect to the associated viewpoint. The collection of views together provide a consistent and complete representation of the system. This

Table 3.1: RM-ODP Viewpoints and focus of concern

Viewpoint	Concern
Enterprise	Deals with the system responsibility, the reason for the system and its use in the environment — purpose and objectives.
Information	Deals with the information that is manipulated in the system and the constraints or rules and relationships that apply to that information — semantics of processing and information.
Computational	Deals with the internal composition of the system and the interactions that take place between components — behaviour, interactions and constraints.
Engineering	Deals with the distribution and configuration of the system and with the mechanism needed to realise the distribution — distribution and infrastructure.
Technology	Deals with the technology and products required to implement the system and with the issues of conformance of the system against its specification — technology, standards, products, code and conformance test points.

notion of views is similar to the definition of views on the Reference Model for Open Distributed Processing (RM-ODP) [Put01, ISO96b, ISO98a].

There are different viewpoint approaches in the literature today. For distributed systems however, the RM-ODP viewpoints are the most popular. According to the RM-ODP, five viewpoints are necessary and sufficient to describe a distributed system (see Figure 3.5): enterprise viewpoint, information viewpoint, computational viewpoint, engineering viewpoint and technology viewpoint. Each viewpoint is a perspective from which one views a system. Table 3.1 summarises the concerns addressed by these viewpoints.

3.3 Development strategies

3.3.1 Early development models

To facilitate the understanding of modern development strategies, we start by looking at traditional development models. In the early stages of system development, designers performed development phases sequentially, i.e., a certain phase could only start after the previous phase had been completed and signed-off. The *waterfall model* [Roy70, Roy87] is the best known example of this development strategy (see Figure 3.6).

Such approaches were mainly based on the assumption that user requirements were fully understood right from the beginning and would not change during the whole development project. This proved to be unrealistic and as a consequence this approach was inadequate for development projects. The weaknesses of this approach were, at that time, mainly attributed to the lacking of feedback between the phases [MJ82]. This led to some extended versions of the waterfall model, in which iterations between the phases were introduced.

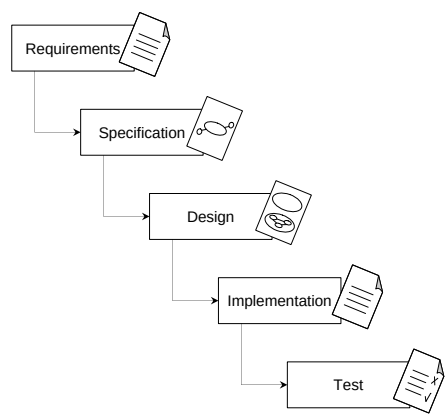
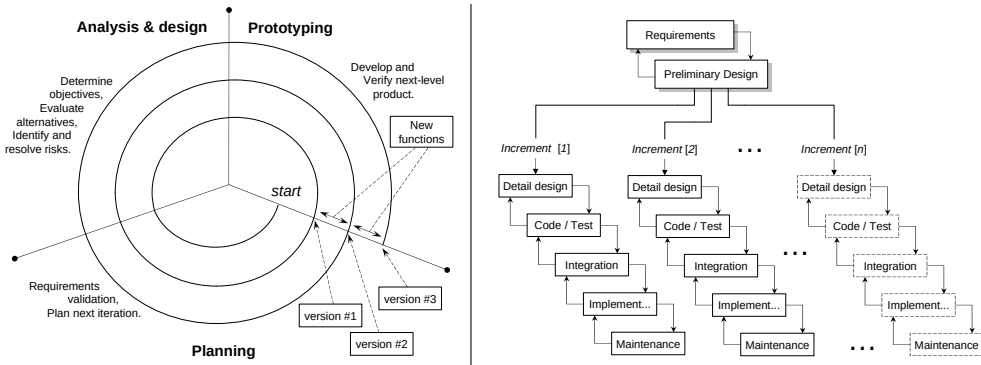


Figure 3.6: The waterfall model

Since it became clear that it was not possible to tie down the user requirements at the early stages of development, designers created methodologies that iterated continuously through the phases. In this case, the development process is executed in small increments or iterations or cycles. Every iteration goes through all the development phases. Initially, a working version of the system is delivered with a reduced amount of functionality. As new iterations are executed additional functionality of the system is considered until the complete



(a) Spiral model

(b) Incremental model

Figure 3.7: Iterative development methods

functionality of the system is realised. Each full iteration employs all of the phases defined in the waterfall model, therefore the milestones of that method remained valid and applicable. Examples of this approach are the spiral model (see Figure 3.7a) [Boe88, BB90], and the incremental model (see Figure 3.7b) [MOL+80].

These iterative strategies were useful for development projects in which the design problem was not fully understood, and particularly if the requirements were expected to change during the life cycle of the project [Boe96]. Advantages of these iterative strategies over the early ones were: they provided some opportunity for incorporating change; and they delivered a working system with a limited functionality at an early stage of development, which allows for faster evaluations. This limited working system allows users to try and test the system leading to possible improvement and clarification of the requirements.

The most important legacy of these early development methods was the identification of important milestones in the development process: the requirements specification, the analysis model, the design model and the implementation. These milestones were obtained at the end of the various development phases. The phases are called requirements, analysis, design, implementation and test, and operation, respectively. These phases are still considered valid today, but the trajectories navigating them in modern strategies are much more complex. From now on, we concentrate on the requirements, analysis and design phases because these are the most relevant for our work.

The objective of the requirements phase is to capture and document the requirements of the system. These requirements are commonly expressed in business terms that describe the problem domain. The requirements phase results in an agreement between the system users and designers concerning the functions, rules and constraints that are visible from the outside of the system. Requirements do not only include functional aspects of the system but also issues such as cost, reliability and development time.

The analysis phase focuses on the definition of the functionality of the system. During this phase, the business requirements obtained in the previous phase are studied to determine what services should be provided by the system. The result of this phase is a specification that prescribes what is the functionality of the system. Such functionality should fulfill the requirements of the users.

The design phase focuses on defining how the functionality of the system is realised. This phase is usually divided into two related stages. In the first stage, the overall structure of the system is specified and all the necessary components are identified. This stage involves the definition of the role of each component and the specification of the interactions between components. In the second stage, the individual components are refined to obtain their internal structure in terms of subcomponents and their interactions. The second stage is applied recursively on the subcomponents until component specifications suitable for implementation have been produced.

3.3.2 Object-oriented development

With the introduction of object-oriented concepts, new approaches to system development were proposed. Designers identified that the ideas of object-oriented programming could be used to strengthen the analysis and design phases of the development process as well. Although most of the practitioners of object-orientation agree that analysis and design are the key activities to ensure success in a development project, there are few others that prefer to write code almost from the start, and work their way through the project using mainly prototyping as their core activity.

In object-oriented development, models at different levels of detail are created during the analysis and design phases. In the analysis phase, a model is created that comprises a number of logical objects. Each logical object in the model corresponds to an object in the real world. These real world objects are often called domain objects. This is because they are only meaningful from the system domain point of view.

In the design phase a definition of the architecture of the system is created. The design model is a definition of the logical objects in enough detail to enable their implementation. In this phase new objects are defined. These objects, also called implementation objects, represent the actual objects to be found in the system implementation as opposed to the logical objects defined before. The design model is a precise description of how the system functions are realised.

Object-oriented development strategies follow a methodology that defines a series of steps or development phases, and the trajectory to navigate through those phases. Some of the best known approaches to object-oriented development can be found in [JCJ92, TP96, Boo94, RBP+91].

Summarising from the different authors we identify the major tasks to undertake during object-oriented development. The steps mentioned bellow are listed sequentially, but they could partly or totally overlap.

1. Gather requirements;
2. Describe typical scenarios;
3. Identify candidate domain objects, determine what these objects are, and what they are responsible for;
4. Establish the relationships between objects;
5. Iterate.

The purpose of steps 1 through 5 is to establish the description of the problem in terms of domain objects and the basic interactions among them.

6. Refine the definition of relationships;
7. Identify any existing or previously implemented objects and generalise them;
8. Refine the objects to determine their internal structure;
9. Iterate.

The purpose of steps 6 through 9 is to create a complete definition of the objects in enough detail to allow their implementation or coding. These steps emphasise the use of controlled iterations to help achieving the required functionality. Iterations are a natural and intuitive way to obtain complete understanding, interpretation and implementation of the user requirements.

The most important approaches that came as a result of the incorporation of object-oriented concepts in system development, were the Objectory process [JCJ92], Object Modelling Technique or OMT [RBP+91] and the Booch Method [Boo94]. The ideas put forward by Booch, Jacobson and Rumbaugh grew and evolve into what eventually became the known as the Unified Process [JBR99].

3.3.3 The Unified Process

The Unified Process [JBR99] is a generic process framework for software development. According to its developers, the Unified Process is more than a single process, but it is a framework that can be specialised for a large class of systems and different application areas. An example of a specialisation of the Unified Process is the Rational Unified Process (RUP) [Kru99].

The Unified Process is organised around three main principles: functional requirements, component-based architecture, and, iterative and incremental development.

First of all, the Unified Process is driven by requirements. The Unified Process defines a systematic and intuitive mechanism for capturing and documenting functional requirements with a focus on value-added to the user. All other activities within the development process can only start once the requirements have been captured and documented.

The Unified Process focuses on architecture. The Unified Process identifies the system's architecture as a primary artefact for conceptualising, constructing, managing

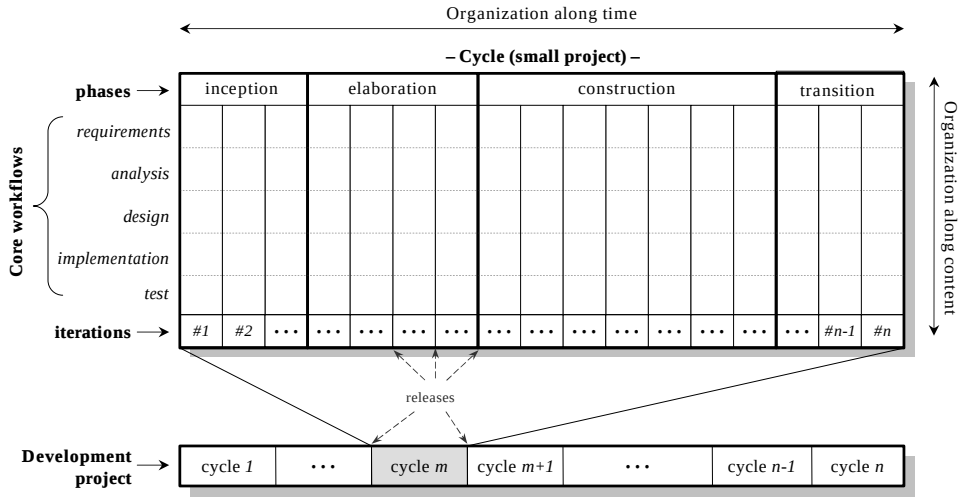


Figure 3.8: The Unified Process life cycle

and evolving the system under development. Aspects of an architecture according to the Unified Process include the selection of structural elements, the specification of the behaviours of these elements and their interfaces, and how these elements work together to progressively compose larger subsystems. Architecture also addresses the issues of performance, scalability, reuse, and constraints either economical or technological.

Another fundamental principle of the Unified Process is its iterative and incremental nature. An iteration is a mini-project that produces a working version of the system. This version is supposed to offer added or improved functionality over the previous version, and that is why the result of an iteration is called an increment.

The Unified Process is organised along two dimensions: time and content (see Figure 3.8). The time dimension represents the dynamic aspects of the development process as it is enacted. This dimension is expressed in terms of cycles, phases and iterations. The content dimension represents the static aspect of the development process or, in other words, the main activities. This dimension is expressed in terms of core workflows.

Within the Unified Process, the life-cycle of a development project is represented as a series of cycles. A cycle ends with the release of a working version of the system to the customers. Each cycle contains four phases: inception, elaboration, construction and transition (see Figure 3.8). Each phase has a specific purpose and concludes with a milestone. At the end of each phase decisions are made about how to proceed with the project.

The purpose of the inception phase is to determine the viability of the proposed

system. The outcome of this phase is a business case, which contains, a.o, the scope of the system, success criteria and risk assessment.

The elaboration phase focuses on determining the ability to build the new system given the constrains faced by the project. This phase results in the determination of functional requirements and the delivery of an architectural baseline that serves as the foundation for further system development.

The goal of the construction phase is to build an operational system according to the decisions made in the previous phases. In the transition phase the working system is incorporated into the user's working environment.

The activities executed during development cut across the four phases (see Figure 3.8). These activities are: requirements, analysis, design, implementation and test. In each one of these activities a blueprint of the system is produced and documented in a number related models, which capture different aspects of the system. These models are the use-case model, the analysis model, the design and deployment models, the implementation model and the test model.

Although the activities and phases are presented here sequentially, they actually lie within an iterative process, and therefore, they are revisited again and again throughout the project life-cycle.

3.3.4 Catalysis

Catalysis [DW99] is a development method that addresses three different aspects of the system, each organised at an abstraction level: the outside (domain/business level), the boundary (component specification), and the inside (component internal design level). Figure 3.9 shows these aspects.

The outside or *domain/business level* is used to identify the users and their needs, modelling the environment where the system has to operate. The boundary or *component specification level* describes the behaviour of the system that is visible from the outside, that is the behaviour of the components that interact with the environment. The inside or *component internal design level* describes how system behaviour is realised in terms of the interactions between components, and it also describes the internal architecture of the components.

Catalysis is based on three modelling concepts: type, collaboration and refinement. Types are used to specify the external behaviour of single objects. Collaborations are used to specify the behaviour of group of objects. Refinement allows the specification of behaviour at different levels of detail. Frameworks describe recurring patterns of these three modelling concepts.

The development of a typical (large) system involves the phases of requirement capturing, system specification, architectural design and component internal design. These phases are performed in accordance with the abstraction levels. Catalysis does not

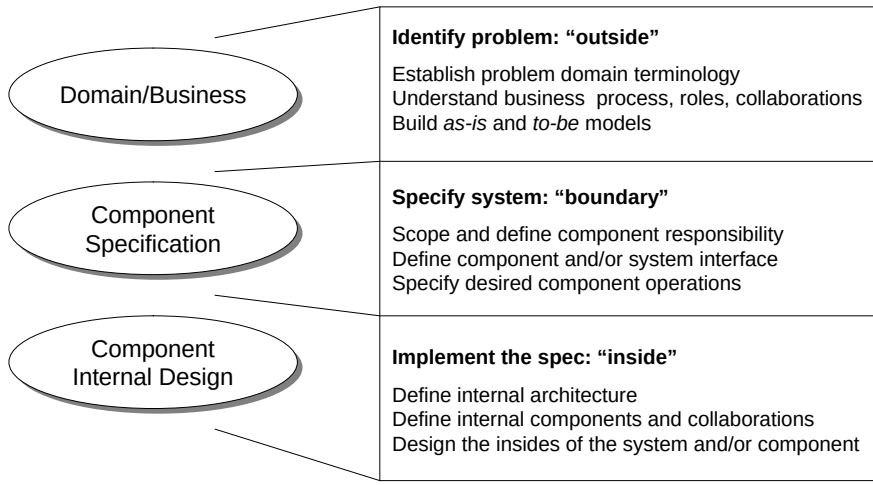


Figure 3.9: The levels of description according to Catalysis

prescribe a unique path through its phases of the development, but it proposes multiple paths through the method, each better suited for projects with certain characteristics.

3.3.5 The Model Driven Architecture

Recently the Object Management Group introduced a new initiative , called the Model Driven Architecture [OMG01b, OMG01a]. The Model Driven Architecture (MDA) is an approach to the full life cycle of systems comprised of software, hardware, humans, and business practices. MDA aims to separate all business or application concerns from the underlying platform technology [OMG03a, KWB03].

The MDA provides a systematic framework to understand, design, operate, and evolve all aspects of such systems, using engineering methods and tools [Bro04]. MDA is based on modelling separately technology-independent and technology-specific aspects of a system, by describing them in separate models. The most important aspect of the MDA approach is the explicit identification of Platform-Independent Models (PIMs) and the flexibility to implement them on different platforms via Platform-Specific Models (PSMs). A platform can be any technology that supports the execution of these models, either directly or after translation into code [D'S01].

From the perspective of system development, a significant quality of the MDA approach is the independence of system specifications (i.e., sets of models) from potential target implementation platforms. A system specification exists independently of any implementation platform and has formal or semi-formal transformation rules onto many possible target platforms [FPSFAA03]. The MDA approach to system (application) specification, portability and interoperability is based on the use of formal

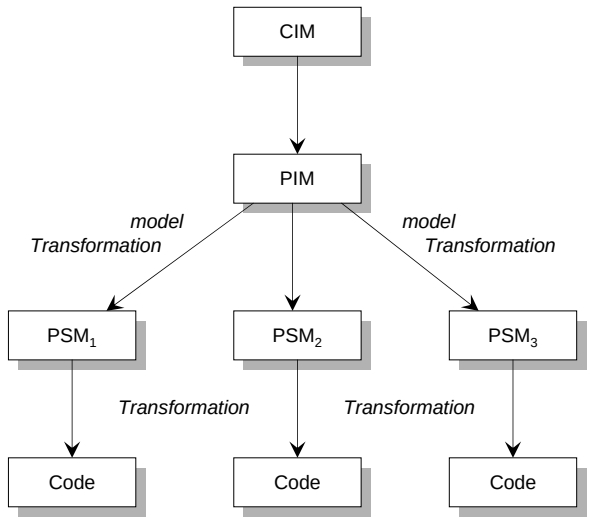


Figure 3.10: The Model Driven Architecture framework

and semi-formal models.

Three different types of model are considered in MDA (see Figure 3.10): computation-independent models, platform-independent models and platform-specific models [OMG03a]. A *computation-independent* model (CIM) focuses on the system environment and its requirements. However, there is no concern for the details of the structure of and processing by the system. A *platform-independent* model (PIM) focuses on the system operation, but hides the details necessary for a particular platform. A *platform-specific* model (PSM) combines the platform-independent model with the details of the use of a specific platform by a system. Model transformation is basically seen as a mapping of elements of one model onto elements of another model.

An essential feature of the MDA framework is to make transformations between models automatic [MB02]. This means that a transformation tool takes a PIM and transforms it into a PSM. A second transformation tool transforms the PSM to code. In this context, a transformation is seen as the automatic generation of a target model from a source model, according to a transformation definition. A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. A transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language.

The MDA defines that first a PIM should be created and then transformed into one or more PSMs. These transformations should be automatically executed by tools if a proper transformation definition has been defined. Designers may benefit because they can focus their efforts on the generation of well-defined, precise, and consistent PIMs that contain enough information about the system, which can be

automatically converted into PSMs for different platforms. Therefore, everything they specify in a PIM becomes completely portable. MDA is made up of a suite of standards that include Unified Modelling Language (UML) [OMG03b]; Meta-Object Facility (MOF) [OMG02]; XML Meta-Data Interchange (XMI) [OMG03c]; and Common Warehouse Meta-model (CWM) [OMG01c].

3.3.6 Conclusion

The strategies described in section 3.3 form a comprehensive list that demonstrates the evolution of system development. We observed that the importance given to iterations over development phases increased as designers try to find ways to manage the impact of the (inevitable) changes in requirements during development.

Apart from the importance of iterations, there is also separation of concerns, which allows one to concentrate on different sets of aspects of the systems at different times along the design trajectory. There is no consent among the various strategies on the sets of concerns. There is consent, however, on the notion that this separation allows the design of systems in a more incremental manner. This notion induces a way of thinking about the system based on distinct and well-defined objectives.

The most current strategies strongly emphasise the role of models. Earlier on, the role of a conceptual model was to guide the implementation and possibly support some future re-engineering activities on the developed system. Nowadays, the role of these conceptual models has been extended as they are seen as a mechanism to express system functionality; they can be interchanged and shared to facilitate interoperability among heterogeneous systems. Furthermore, model-driven approaches promote the independence of the application model (system specification) from the implementation technology and platform. Therefore, an application model can be developed independently from any implementation model and may have a formal mapping to many alternative platform infrastructures.

We adhere to these last two principles, namely the separation of concerns and the model-driven approach, to devise our methodology in support of GSI system design.

Chapter 4

Geo-services design methodology (GSDM)

The significant problems we face cannot be solved
at the same level of thinking we were at when we created them.

Albert Einstein

This chapter introduces the Geo-Services Design Methodology (GSDM) which is tailored to the development of geo-information systems. The chapter starts by describing the main phases of GSDM. The chapter also describes the role played by design models in GSDM and motivates the need for a metamodel. Next, the chapter introduces a metamodel for GSI services, and finally the chapter discusses the main architectural elements of the methodology.

The structure of the chapter is organised as follows: section 4.1 gives a general introduction of GSDM; section 4.2 illustrates the scope of GSDM in the context of the development process and discusses the main phases of GSDM; section 4.3 explains the role played by models in GSDM; section 4.4 introduces a metamodel for GSI service models; and finally section 4.5 discusses the main architectural elements used in the description of GSI components.

4.1 Introducing GSDM

The trajectory followed during the construction of a system should start with a conceptual modelling phase, during which models that describe static and behavioural aspects of the system are produced [Gra01]. Traditionally, in the design of geo-information systems these two aspects have not received the same amount of attention,

since little or no focus has been put on the behavioural aspects of these systems.

Conceptual design should be supported by a development methodology. A development methodology provides a set of structured procedures and rules that guide a designer in the process of designing a system (see Section 1.4).

In this research we have developed a methodology to guide the design of distributed geo-information services, which we call the Geo-Service Design Methodology or GSDM for short. GSDM is organised around the concepts for behaviour modelling supported by ISDL [QFPS02]. Although the methodology can be used to design distributed systems in general, we concentrate on its application for defining both structural and behavioural aspects of distributed geo-information systems.

GSDM focuses on the definition of the services provided by GSI systems (see Section 2.7). A service is a collection of functions or operations organised in a way that they exhibit a behaviour of value to a user. The functions used within a service are provided by independent entities, and these functions are available at different system nodes. Such services have to be formally defined before they can be properly implemented. The internal structure of the service (i.e., the service realisation) describes how different components interact to provide the desired service.

4.2 GSDM overview and scope

When we choose to regard some part of the 'real world' as a system, then we are immediately defining that part as one entity, and at the same time, indirectly we draw a line (the system boundary) between what should, and what should not, be considered part of the system.

The boundary of the system does not simply separate what is part of the system and what is not. It also represents the point of contact between the system and the outside world. It is at the system boundary that interactions take place.

It is not possible, however, to ignore everything that exists outside the system boundary. On the contrary, we must identify the things that are not themselves part of the system, but affect and/or shape the system's operation. For example, if we define a national mapping organisation as a system, then we have to recognise that such organisation has close relationship with entities like image suppliers or budget providers like, e.g., the government.

The determination of the system boundary does not only depend on the identification of the system itself, but also in the determination of that system's purpose. For example, if we choose to consider a prison as a system to, say, 'rehabilitate criminals', then the system includes entities such as therapist and social workers, whereas if we decide to consider a prison as a system just to 'punish criminals', then such entities would not be included within the system.

Similarly, If one chooses to regard a map, as it is conventionally done, as a system for, say 'representation of information' then one has to deal precisely with a single scale and strict use of location and orientation (e.g., a topographic map), whereas if one decides to consider a map as a system for 'communication', say, a map of an underground system, then one can use multiple or distorted scales, relative locations and other types of spatial distortions that facilitate communication with the map users.

These considerations lead us to the identification of four main areas of interest with respect to a system: the objective or purpose (the contribution of the system to the surrounding environment in terms of what it does), the boundary (the interfaces of the system), the outside (the environment in which the system operates), and the inside (the system's internal structure or structures and composition). Our methodology has been organised according to these levels of concern.

To address these concerns levels, GSDM defines two main system perspectives that enable the representation of a high-level architecture of a GSI system:

- the *external perspective model* captures information about the purpose and environment of the system in service specifications. This includes the service definition, the boundary level interactions through which the service can be accessed and the entities that participate in these interactions;
- the *internal perspective model* captures information about the system's architecture. This includes decomposed service specifications that in turn comprise identification of architectural elements and definition of behaviours. This behaviour definitions describe the interactions, between architectural elements, that are required to realise the system's services.

The external perspective aims at identifying and explicitly delimiting the scope of the system under development and helps determining the objectives of the development process. It is at this stage that the boundary between the system and its environment is defined. The external perspective involves: the organisation and understanding of requirements; the specification of system functionality or system responsibilities (services), which represent the contribution of the system to the environment; and the definition of interactions that prescribe how external entities may interact with the system in order to access its services.

The internal perspective aims at describing the internal system structure in terms of compositions of simpler or more elementary architectural elements. Arrangements of these architectural elements form the so called decomposed service specifications. A decomposed service specification identifies a group of architectural elements and describes how these elements interact to perform a system function (service). Many different alternative compositions are allowed that implement the same functionality in a different way.

The internal perspective actually covers two related concerns, namely the set of ar-

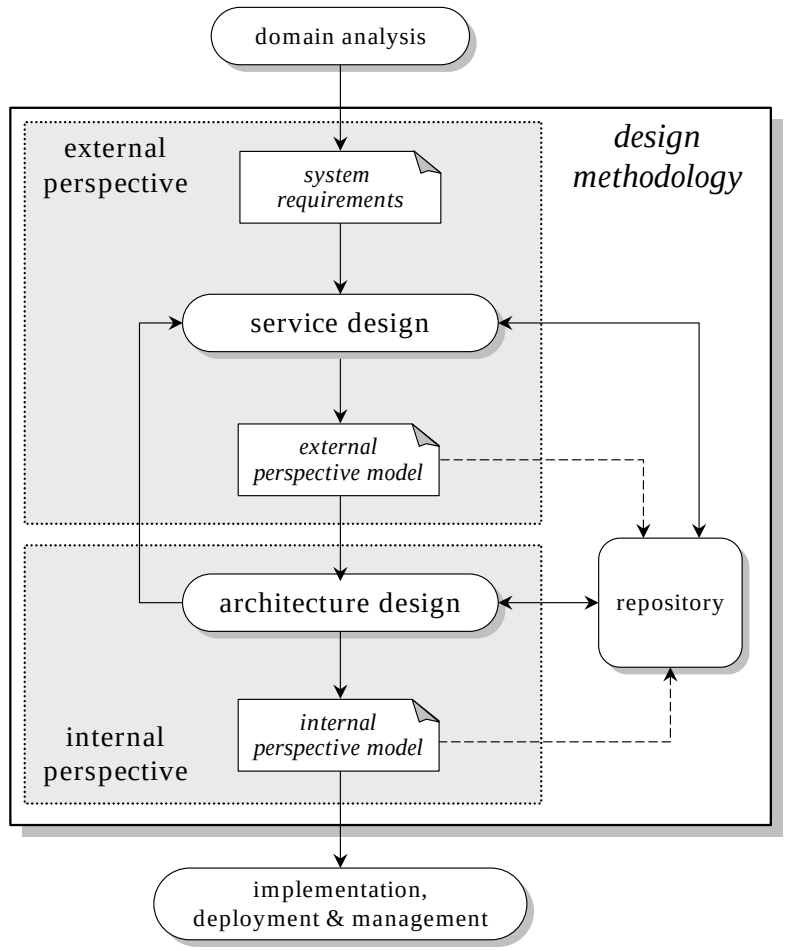


Figure 4.1: Main phases and scope of GSDM

chitectural elements and how they interact to realise system's services. Therefore, to obtain the internal perspective model initially one is forced to consider each individual potential element separately. This separate consideration is essential when determining the suitability of an element for its participation in a service realisation.

The description of individual architectural elements includes: the mechanisms by which the element can be accessed or instantiated, which in turn determines how architectural elements can connect to each other; the internal structure of each architectural element; and, the behaviour exhibited by the element when applicable.

Once models or specifications of services and architectural elements are created, they are stored in a repository. Models stored in the repository can be used by designers

in other arrangements to realise other more complex services. Figure 4.1 shows the relationships between the perspectives of GSDM and the phases of development, and how they influence each other.

4.3 The role of models

Our approach to geo-information services design focuses on the use of conceptual models as an intermediate step in the development process, which sit in between requirements and the actual implementation (see figure 4.1). The purpose of this step is to obtain a conceptual description also called abstract specification of every system part and every system function (internal or external). This is done solely to enable and facilitate reuse and to enhance flexibility.

The main benefit of these models is to serve as the basis for the specification of complex services. If a model properly describes an architectural element, that is, with the relevant information at the correct level of detail to enable one to determine what it does and how to access the function it provides, then this architectural element can be easily reused. By reuse of architectural elements, we mean the inclusion of previously designed element in multiple service definitions. We use constraint-oriented composition to combine multiple architectural elements (see section 5.4.2). This way

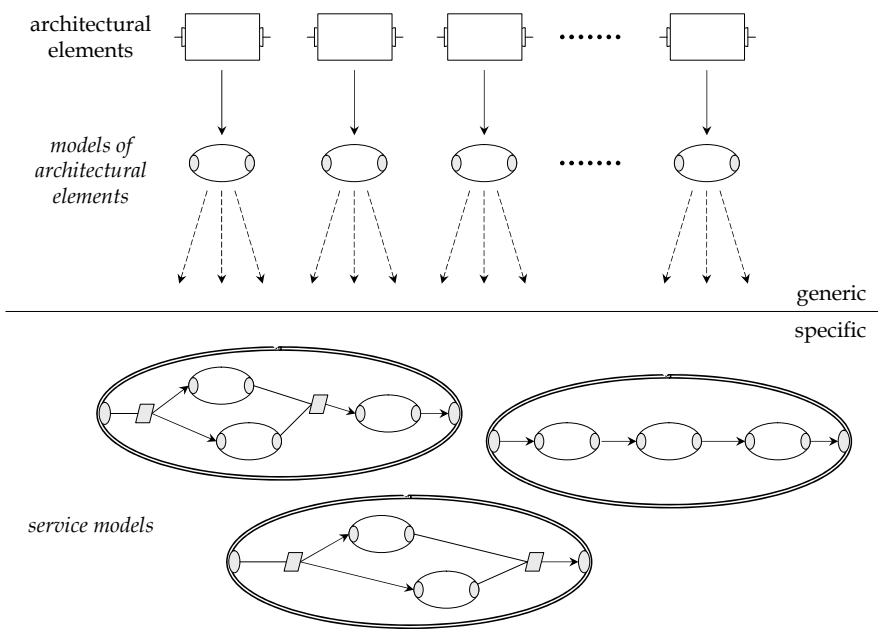


Figure 4.2: Architectural elements, element models and service models

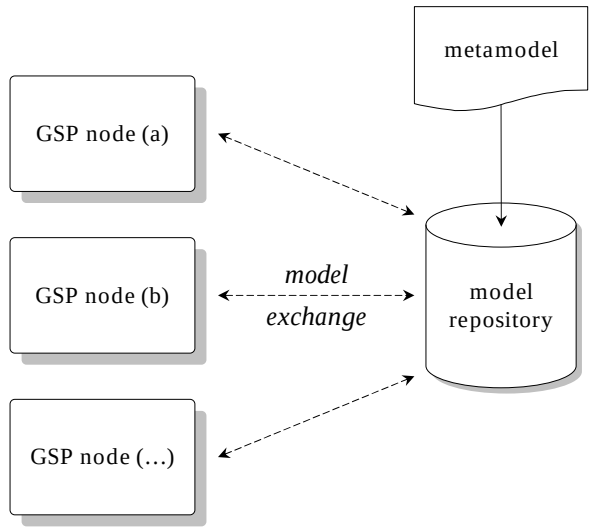


Figure 4.3: Role of the metamodel in the GSI architecture

it is possible to assemble large processing chains by correctly combining models of (generic) architectural elements to form more specialised specifications (service models) with the intention of provide more specific or particular services (see Figure 4.2).

Since the models of architectural elements prescribe the behaviour exhibited by individual elements, a service model can be used to choreograph the realisation of the service specified in the model. Additionally, once a service model is available it can itself be reused, as a generic element, in another definition as a part of a yet more specialised service. Reuse and flexibility is ultimately achieved by developing, sharing and integrating well-defined models of all composing elements of the system.

For this approach to work, models need not only to be interchanged between participants, but they also have to be understood by all parties involved. This can only be achieved if the models are based on the same metamodel. Such metamodel should therefore provide a rigourous abstract syntax for defining models. Figure 4.3 shows the role of the metamodel in the GSI architecture. The metamodel enables the implementation of a repository where compliant models of GSI services can be stored. Hence, the repository becomes the central component in a GSI system. The repository supports the exchange of models between different service providers, thereby facilitating the use of these models in combinations to form more complex service models that address specialised sets of requirements.

A metamodel defines the set of design concepts and their relationships, which one can use to produce models of some system according to a specific objective. A design concept, also called modelling concept, is a building block that can be used in the construction of a model. A design concept represents one or more related properties

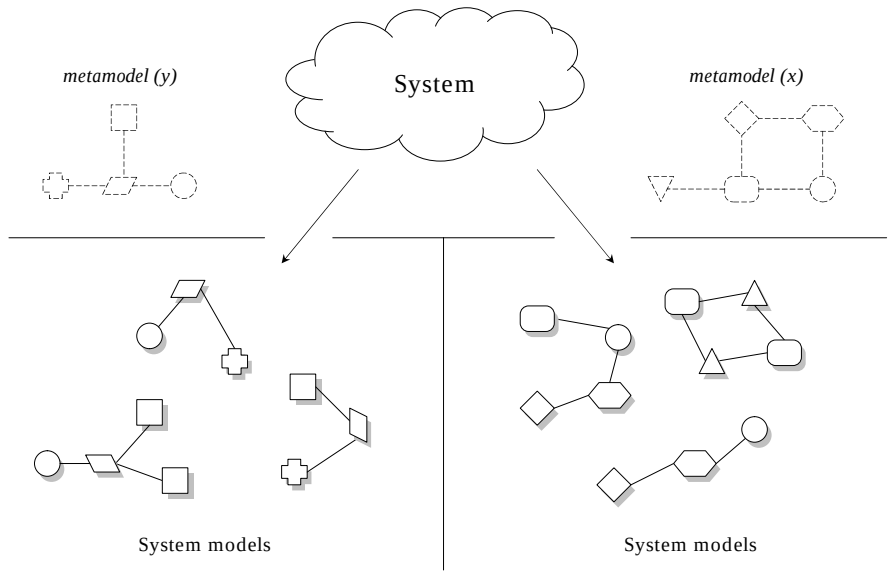


Figure 4.4: Metamodels and system models

of a system or system part that are considered relevant in the design of a system. The complete collection of design concepts should allow one to model all relevant system properties. Relationships between design concepts define the possible ways in which a model can be constructed from of these concepts.

Figure 4.4 illustrates the relations between system models and metamodels. In Figure 4.4 design concepts defined in a metamodel are represented by different geometrical shapes, while a relationship between concepts is represented by a line connecting two concepts.

Which metamodel should be used depends on the modelling needs of every particular project. For the purpose of designing GSI systems, we require design concepts suitable for specifying the system's functionality and its environment, the system's internal structure in terms of its composing parts or subsystems and their relationships, and the contribution of each part to the system's overall functionality. This translates into the following modelling needs:

- to represent the system, its logical or physical parts and any external thing that interacts with the system, which could be a person, another system, etc., as single entities capable of exhibiting behaviour;
- to represent the locations where interactions between different entities occur;
- to represent behaviour according to different related abstraction levels;
- to be able to discriminate between the behaviour and the entity that carries the

behaviour, such that an entity could potentially exhibit multiple behaviours.

- to be able to structure behaviour into units behaviour and their relationships;
- to represent anything that is used or produced in a behaviour.

Additionally, the selection of an appropriate set of design concepts should adhere to the following quality principles [Sin95]:

- *consistency*, which requires that concepts should be consistent in their representation of the aspects in the real world;
- *orthogonality*, which requires that distinct concepts should be used to represent different aspects;
- *propriety*, which requires that concepts should be proper to the modelling needs;
- *generality*, which requires that concepts should be of general purpose in a given domain and the complete set of concepts be sufficient to cover the needs of the domain.

4.4 Metamodel for GSI

The types of service provided by a GSI system spread along the whole geo-information value chain. This value chain starts with identification of information sources that are used in different geo-processing tasks to create value added geo-information products. These products are subsequently used in various types of analysis with the purpose of deriving new information that is not directly obtainable from the sources. In most of the cases diverse combinations of tasks and information sources are required to solve specific problems and help user communities to make sense of the geographical world that surrounds them.

The structural organisation of this work is consequently formed by arrangements of independent functions and data sources organised in such way that large geo-processing tasks can be accomplished. Any of these arrangements defines a specific behaviour. This behaviour is accomplished through the creation, manipulation or transformation of some items or artefacts, and must be carried out by entities within the system.

According to these criteria and to the modelling needs mentioned in the previous section, we need a metamodel to be able to build repositories where our models can be properly, structurally and consistently stored and retrieved. We introduce a metamodel to be used in the creation of repositories to store our models (see Figure 4.5). This metamodel is a specialisation of the ISDL metamodel [Qua03]. The metamodel is organised in a number of classes, where each class addresses a group of related design concepts.

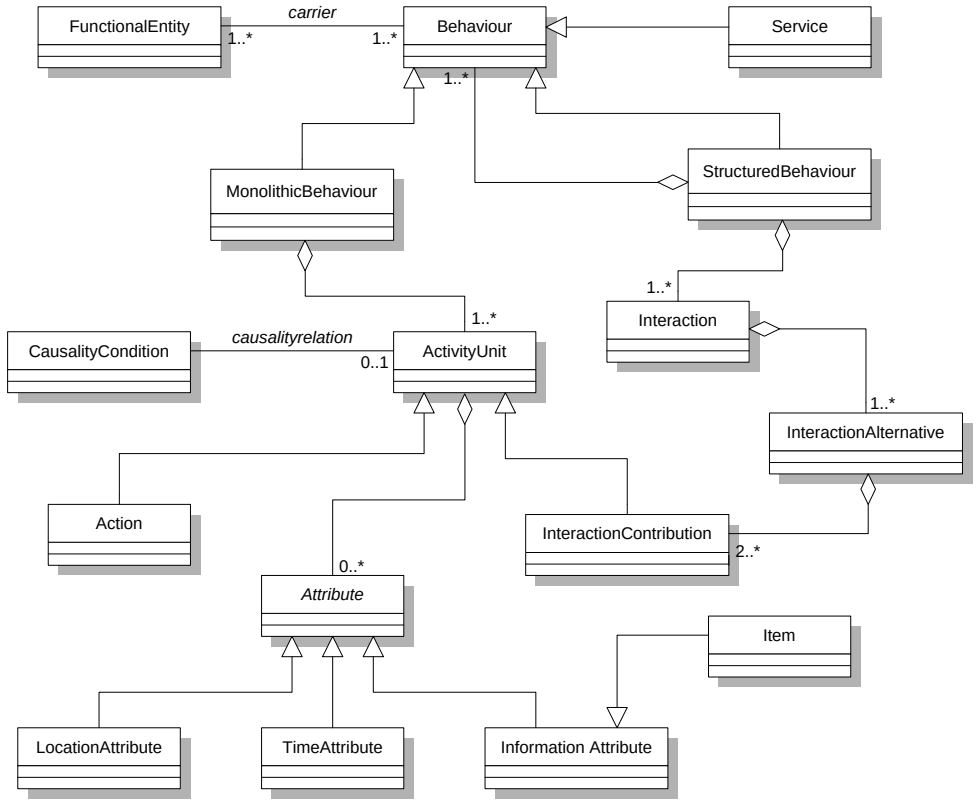


Figure 4.5: A metamodel for GSI services

The abstract class *behaviour* defines the behaviour concept, which models some type of system behaviour. A behaviour is a (partial) description of the system that describes a distinct part of that system functionality.

An instantiation of a behaviour results in a *service* of value to a user. Multiple behaviours may provide the same service. Behaviours are associated with functional entities. *Functional entities* are capable of exhibiting the characteristics defined in a behaviour. A functional entity represents a logical or physical part of the system capable of executing behaviour in the real world. A functional entity executes behaviour by itself or in cooperation with other functional entities. A behaviour can be carried by more than one functional entity. A functional entity can exhibit more than one behaviour.

Two sorts of behaviour types are distinguished: *structured behaviour*, which are compositions of one or more related smaller behaviour; and *monolithic behaviour*, which are not further decomposed into smaller behaviours.

A monolithic behaviour consists of a group of related *activity units* that can take the

form of actions or interaction contributions. An activity unit represents an atomic piece of work at a given abstraction level.

An *action* represents an activity that is defined entirely within a single behaviour. An *interaction* represents an activity in which two or more declared behaviours participate (cooperate). An *interaction contribution* represents the participation of a behaviour in some interaction.

At higher levels in the behaviour hierarchy, the participation of one or more monolithic behaviours in some interaction may be represented by the participation of the structured behaviour in which these monolithic behaviours are defined as sub-behaviours. This may be applied recursively to structured behaviours that are defined as sub-behaviours, such that the participation of a structured behaviour in some interaction may represent the participation of monolithic and structured sub-behaviours.

Alternative groups of sub-behaviours may participate in an interaction. Therefore, an interaction is defined as a set of one or more *interaction alternatives*, where each alternative represents an optional group of participating sub-behaviours.

A *causality condition* is associated with each activity unit. This association is called a causality relation. A causality condition defines the type of relation between two or more activity units. This relation is used to specify how the occurrence or execution of activity units depends on the occurrence or non-occurrence of other activity units.

The completion of an activity produces some result that can be manipulated by other activities. An activity unit can have attributes. These attributes represent the result that is established by an activity unit.

Three attributes are defined:

- the *information attribute*, which represents the product (typically some information) that has been produced by the activity unit;
- the *time attribute*, which represents the time moment at which the product is available;
- the *location attribute*, which represents the location where the product is available.

Items are a special class of the information attribute. Items represent the information that is directly manipulated by an activity unit. The type of manipulation that can be performed on an item by an activity unit are create, use, change or destroy the item.

The metamodel described here only allows for the definition of an abstract syntax for the design concepts used in the creation of GSI models. The semantics of the concepts is provided by the ISDL modelling language (see Chapters 5). For further detail on the ISDL metamodel, we refer to [Qua03].

4.5 Architectural elements

To deal with the issues concerning the internal structure of a service, we have to identify and characterise the architectural elements within a distributed geo-information system. First of all we ignore details of implementation and communication of these elements in order to focus on the functions they provide and the constraints that apply to their interactions with one another.

Architectural elements basically encompass the different components of the system, however, here we refer to a component in a slightly different manner than the way this term is used by the software engineering community. The term component in software engineering disciplines refers to a self-contained unit of independent deployment, with well-defined interfaces that has no persistent state [Szy02]. Usually, a component provides a particular function or group of related functions. A component is a reusable unit of composition that can be used to form applications with other components in the same or other computers in a distributed network. In this thesis, whenever we refer to this type of component we explicitly say *software component*.

We refer to a component not only to represent units of software, but in general to any architectural element of the system that provides a function required in a larger processing chain. Such processing chain is formally described in a service realisation. Components in this context can thus be used, for example, to refer to some abstract representations of data. or to an action or set of actions performed by a human, and that may yield a necessary result or provide a required function. In order to create abstract representations of components of geo-information systems, GSDM makes use of architectural elements. GSDM ignores the details of component implementation to focus on the roles of components, the constraints upon their interaction with other components, and their use of data.

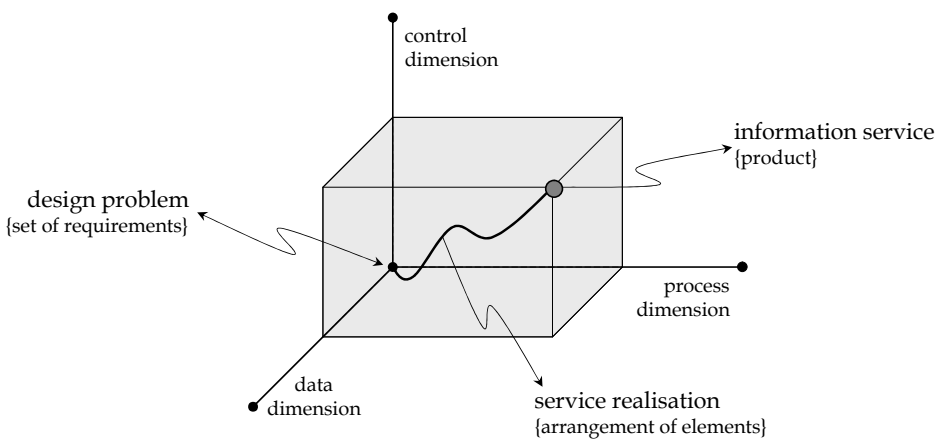


Figure 4.6: Modelling dimensions

We distinguish three different types of architectural elements within a GSI system (see Figure 4.6):

- data elements;
- processing elements; and
- connecting elements.

The *data elements* represent the information that is used, manipulated and/or generated by the system. The *process elements* represent the geo-processing capabilities of the GSI system, which can perform transformations on data elements. The *connecting elements* are like mediators, they represent the relationships between other elements. The connecting elements represent the conditions and constraints that define how the different elements may interact and how they are organised with respect to each other in a service specification. A similar approach to system elements can be found in [PW92, Fie00].

To illustrate the differences between these elements we can use metaphorical example. Consider the games of soccer and handball. The two games are similar in structure, they use a ball as data element and players as processing elements. The difference between them lies in how these elements are allowed to interact with each other, which are the context and rules of the games (the connecting elements). These connecting elements are defined by game designers based on what they want to achieve with the interactions.

Separation of concerns is the principle behind this modelling dimensions. For example, by separating the concerns on the nature and state of data from the data processing concerns, we simplify the processing elements allowing them to change and evolve independently. In the same way portability of the data is improved by avoiding that the data remain encapsulated and hidden within processing elements. However, the drawback is that we lose the advantages of information hiding and therefore a mechanism is required for processing elements to identify and understand the data types.

The origin of the reference system determined by the modelling dimensions in Figure 4.6 represents the starting point of a development process. The process is activated by a design problem. Design problems represent the various inputs (requests) to the GSI system that have to be converted into services. A service has to be specified before it can be realised. The path taken through the cube shown in Figure 4.6 corresponds to the functional specification of a service that satisfies a specific design problem using an appropriate arrangement of components. The execution of a predefined service specification generates the desired service.

Chapter 5

Design concepts

If you don't ask why this? often enough,
somebody will ask why you?

Tom Hirshfield

In this chapter we introduce the design concepts and their combination rules necessary to represent architectures of distributed systems. An architecture consists of a structure of parts and their functionality (functions). This chapter introduces the concepts of entity and behaviour, and the other concepts that form our basic design model for distributed systems. An entity is an abstract concept that can be used to represent systems and system parts, while a behaviour is an abstract concept that can be used to represent the functions of these systems and system parts.

This chapter is structured as follows: section 5.1 introduces the design concepts necessary to represent structures of interconnected entities, section 5.2 introduces the design concepts necessary to represent behaviours, section 5.3 introduces decomposition as a tool to produce designs at lower levels abstraction, and finally, section 5.4 discusses the structuring of behaviours as compositions of sub-behaviours.

5.1 Entity structures

A system can be modelled as a structure of interacting entities, each one of them representing a logical or physical part of the system. This section presents the concepts necessary to represent such structures.

5.1.1 Entities

An *entity* is an abstract concept that models the identity of some system in the real-world (either existing or being built). At a certain abstraction level a single entity may be used to describe, for example, a complete system. At another abstraction level, though, the same system may be represented as an arrangement of interacting entities. The entity concept is useful in systems development, because the concept makes it possible to structure systems as compositions of entities, which facilitates insight. This system structure can be made of either physical or logical parts or both.

A system has certain properties, which can be represented as characteristics of its corresponding entity. For example, a car can be modelled as an entity that is made of a particular material, with a particular form, and with a particular colour. The material, the form, and the colour are the properties of the car that in this case can be associated with its corresponding entity.

Geo-information systems have specific characteristics as well. The most important characteristic of a geo-information system, in the context of this research, is its functionality. This characteristic can be represented in terms of behaviour (see section 5.2). This means that to properly model the way a geo-information system operates, a behaviour has to be associated to the entity that represents the system. To distinguish between different entities, each entity has to be uniquely identified using an entity identifier (entity names).

5.1.2 Interaction points

Users interact with systems through mechanisms that are specially designed to support these interactions. An *interaction point* models the mechanisms through which an entity can interact with its environment. Interaction points serve to delimit an entity and hence to separate this entity from its environment. An entity without interaction points is meaningless for its environment. Two or more entities must share an interaction point, to enable their interaction.

Users of geo-information systems interact with these systems through, for example, user interfaces. We considered an interface as the shared mechanism that support

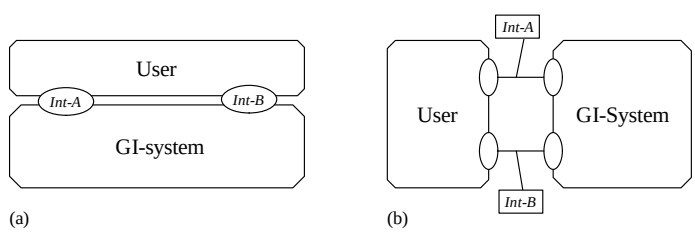


Figure 5.1: Entities with shared interaction points

the interaction between the system and a user. This mechanism can be modelled as the interaction point that is common to the geo-information system and to the user. Analogously to entities, interaction points must also be uniquely identified, for instance, by means of an interaction point identifier.

Figure 5.1a shows the graphical representation of entities and interaction points. An entity is represented by a non-overlapping polygon with cut-off corners. An interaction point is represented by an oval that overlaps with the entities that share this interaction point. The identifiers of entities and interaction points can be placed either inside their graphical representations or in text boxes linked to the corresponding entities or interaction points. Figure 5.1b depicts an alternative representation.

5.2 Behaviour concepts

A system is designed to provide a specific functionality, which makes it useful to its environment. An entity, as explained in the preceding section, allows to define what the system is. In order to specify what the system does, we associate the entity with a behaviour. A *behaviour* represents the functionality of the system modelled by its associated entity.

The functionality of a system consists of all the activities performed by the system and the relationships between these activities. We use behaviour models to represent the functionality of a system. A behaviour model is created by combining design concepts that represent activities performed by the system and their relationships. In this section we introduce the concepts required to specify behaviour.

5.2.1 Actions

An *action* is an abstract concept that represents a real world activity. The occurrence of an action represents that its corresponding activity has been successfully performed. Consequently, an action models the result (product) of an activity.

Examples of activities are filling up a request form, extracting data from a data repository, deriving the slope of the terrain, measuring the area of a parcel or identifying a set of land use classes from a satellite image. These examples show the variety of activities in geo-information production in terms of required expertise or machinery, complexity, time duration, etc.

Each action is uniquely identified and is the most abstract representation of an activity: it models only the result of an activity. As a consequence, an action cannot be considered as a composition of other actions at the abstraction level at which it is defined. This is called the atomicity property of actions.

To illustrate this, consider the activity of producing a landuse map. Obviously, this

activity takes time and involves, e.g., visiting the area of interest to collect some ground truth, acquiring satellite images, classifying the images, etc. However, if only map production is relevant, then we can model this activity with a single action, *produce landuse map*, which models this result. The action occurs at the moment the map is finished, which is a single, indivisible point of time. An action is therefore an indivisible unit of activity (atomic) at a certain abstraction level.

The atomicity property of does not imply that actions cannot be decomposed into sub-actions or actions at a lower abstraction level. Action decomposition is needed whenever the activity represented by an action has to be defined in more detail (see section 5.3.2). For example, it is perfectly allowed to decompose the action *produce landuse map* into the actions *collect ground truth*, *classify images*, etc. In the later case, however, one is considering the original action at a lower abstraction level.

The atomicity of actions has the following consequences:

- An action either occurs completely, or does not occur at all. In contrast to activities, actions cannot occur partly;
- If an activity delivers multiple results at multiple points of time and the difference between these results is relevant, then the activity has to be modelled as multiple actions.

Activities can yield different forms of results. These results could be a concrete product (a map, a land title, a book), information (personal data, a message) or a service (a geo-referenced image). Furthermore, the result of an activity, once produced, becomes available at a particular place and at a specific point in time.

An action has attributes that can be used to represent the most relevant characteristics of the activity it represents, namely the result produced by the activity, the time moment when the result becomes available, and the physical or logical location where the result can be found. Therefore an action has three attributes:

- the *information attribute* models the result of an activity;
- the *location attribute* models the physical or logical location at which the result of the activity is made available;
- the *time attribute* models the point in time at which the established result becomes available.

All the mentioned characteristics of activities are modelled in terms of values of the different action attributes. Action attributes represent the characteristics of an activity that are of interest to other activities once this activity has been successfully completed. An action models what result is established in an activity, when and where, but not how this result is established. An action abstracts from all mechanisms of the activity that make it possible for the activity to reach its result.

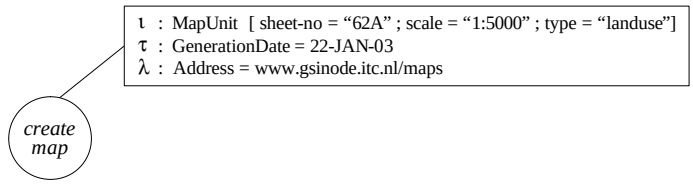


Figure 5.2: Action representation

We graphically represent an action by a circle. The action name is placed either inside the circle, or in a text-box connected to the circle through a line. Action attributes are represented in a text-box connected to the action circle. The values of the information, time, and location attributes are represented by the symbols ι, τ, and λ, respectively.

Figure 5.2 depicts action *create_map*, which represents the activity of creating a map. In Figure 5.2, action *create_map* models the generation of the landuse map, No. "62A" on January 22, 2003, which has been produced at www.gsinode.itc.nl/maps.

Actions are design concepts for defining models that represent the behaviours of systems. Since a model prescribes the implementation of a system, the mechanism that implements an action must be reliable, such that behaviours can be properly implemented. Thus, if the activities specified in a model are to be carried out by, for example, people or computers, the model should be structured in terms of such entities. The actual physical assignment may occur at the end.

5.2.2 Interactions

Some activities in real-life are performed by two or more systems in cooperation. In that case such activities take place at a shared interaction point of the entities that represent the systems. An *interaction* is an abstract concept introduced to model an activity performed by multiple systems in cooperation. The involvement or participation of an entity in an interaction is called an *interaction contribution*.

Examples of interactions are transferring a product (e.g., from a seller to a buyer), logging in to a system (usually a computer system and a user), or posting a letter, which requires the co-operation between the person posting the letter and the mail box (or, more abstractly, the postal service).

Interactions have the same attributes as actions. However, when two or more systems interact, each of them has interaction constraints that define under which conditions they can participate in the interaction and which are the results that can be established by the activity. Therefore, two or more systems can only interact if the intersection of their constraints on the possible values that can be established in the interaction attributes does not result in an empty set of values. This means that after applying all constraints at least one valid information value must be established. This

applies to all three attributes information, time and location. Three basic forms of value establishment are distinguished:

- *value checking*, which represents that one behaviour establishes a specific value x, while other behaviour requires a specific value y. In order to allow the interaction to happen the following condition must hold: $x = y$;
- *value passing*, which represents that one behaviour allows a specific value x to be established, while other behaviour accepts any value from a set of values Y. In order to allow the interaction to happen the following condition must hold: $x \in Y$;
- *value generation*, which represents that one behaviour allows any value from a set of values X to be established, while other behaviour allows any value from a set of values Y to be established. In order to allow the interaction to happen the following condition must hold: $X \cap Y \neq \emptyset$.

Similarly to actions, interactions are also uniquely identified by means of interaction identifiers. The graphical representation of an interaction stresses the partitioning of responsibilities between the participating entities. An interaction is represented using connected circle segments, one circle segment for each participating entities, Each circle segment represents the interaction contribution of the participating entity.

Figure 5.3 shows an interaction that models a login operation as an activity performed by two systems: the computer system and a human user. The interaction contribution of an entity is identified by the interaction identifier and (optionally) the entity identifier, separated by a dot. Interaction identifiers are underlined in order to distinguish them from action identifiers.

In Figure 5.3, interaction contribution login.user (system user) defines that the user would login at any time with the password "pjf286" to the address ftp.gsi.sys. interaction contribution login.user (computer system) defines that the system accepts users to login to the address ftp.gsi.sys, with one of two passwords "pjf286" or "vmd404", somewhere between 07:30 and 20:30 hours every day (no constraint on the day has been defined).

This example illustrates value checking, since both for the user and for the computer system the same address value must be established for the interaction to happen. The example illustrates value passing, since the user logs in with the password "pjf286"

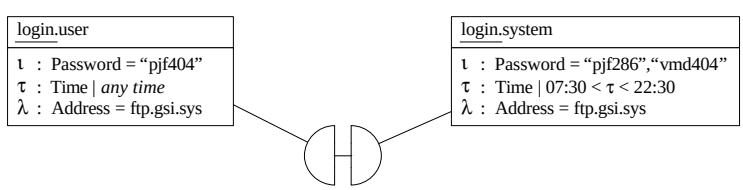


Figure 5.3: Interaction representation

and the computer system allows two possible passwords for a login operation, one of which matches the users password. This example also illustrates value generation, as the user may login at any time and the system allows login operation during the specific time period 07:30..22:30. Therefore the interaction is only allowed to occur at an allowed time moment.

5.2.3 Causality relations

In general, the occurrence of an action depends on the satisfaction of a condition. This condition consists of some statement on the occurrence or non-occurrence of other actions. For example, the development of a set of aerial photographs depends on these photographs being taken before. Once the condition 'the aerial photos have been taken' is satisfied, the development photographs action becomes possible, and may happen at any time. Some actions do not depend on other actions, and are allowed to occur at any time. These actions are called *initial actions*.

Causality relations are used to model the relationships between actions (interactions). A *causality relation* defines how the occurrence of an action, the so-called *target action*, depends on the occurrences or non-occurrences of other actions. A causality relation models the conditions under which an action becomes enabled. A causality relation consists of:

- a *causality condition*, which defines how the occurrence of the result action depends on the occurrences or non-occurrences of other actions;
- *action attribute constraints*, which define how the values established in the information, time and location attributes of an action influence the occurrence of a related action (a target action) and possibly the values of its information, time and location attributes; and
- a *probability attribute*, which defines the likelihood of occurrence of the target action in case the causality condition and action attribute constraints are satisfied.

Causality conditions

Consider two actions a and b, which happen at time moments τ_a and τ_b respectively. There exist three possibilities in terms of the temporal ordering of the occurrence of these two actions: a occurs before b ($\tau_a < \tau_b$), a occurs at the same time as b ($\tau_a = \tau_b$) or a occurs after b ($\tau_a > \tau_b$). These possibilities do not overlap, therefore we can define some basic causality conditions for action a as follows (a causality condition is represented by the symbol 'γ'):

- $\gamma = b$, defines that the occurrence of action b is a condition for the occurrence of action a, such that the occurrence of b must precede the occurrence of a.

Since action a is only allowed to occur after action b occurs, we say that the occurrence of action b enables the occurrence of action a ($\tau_a > \tau_b$). This condition is called a *enabling condition* and action b is called a *enabling action*.

- $\gamma = \neg b$, defines that the non-occurrence of action b is a condition for the occurrence of action a, until a has occurred. Action a is allowed to occur when b does not occur before a and b does not occur simultaneously with a. Since action a can only occur if action b has not occur, we say that the occurrence of action b disables the occurrence of action a ($\tau_a < \tau_b$). This condition is called a *disabling condition* and action b is called a *disabling action*.
- $\gamma = = b$, defines that the occurrence of action b is a condition for the occurrence of action a, such that the occurrence of b must happen simultaneously with the occurrence of a ($\tau_a = \tau_b$). This condition is called a *synchronisation condition* and actions a and b are called *synchronised actions*, since their occurrence have to be synchronised.
- $\gamma = \surd$, defines that action a does not depend on the occurrence of other actions, such that action a may occur independently from other actions at any time. This condition is called *start condition* since no condition exists, and action a is called *initial action*.

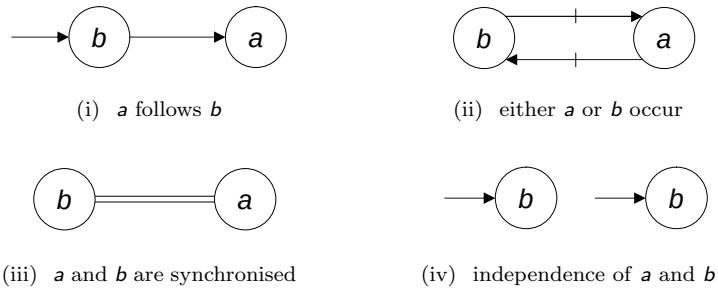
Simple behaviour definitions

A behaviour definition consists of one or more actions and their relations. Since we use causality relations to model the relations between actions, we define a behaviour in terms of a set of causality relations, one for each action of the behaviour.

The textual representation of causality relation has the following form: $\gamma \rightarrow a$. Where a is the identifier of the target action, and γ is the causality condition of action a.

The textual notation of a behaviour definition consists of a behaviour identifier, followed by the symbol '=', and a set of causality relations delimited by the symbols '{' and '}'. For example, the notation $\mathsf{B} = \{\surd \rightarrow d, \surd \rightarrow c\}$ represents a behaviour definition named B, which contains two independent actions c and d. Some examples of simple behaviours are explained below, and are depicted in Figure 5.4:

- a follows b: a enabling condition is used to define the sequential ordering of a and b such that a occurs after b. The textual notation for this behaviour is $\mathsf{B} = \{\surd \rightarrow b, b \rightarrow a\}$. The enabling condition relation is graphically represented by a solid arrow from b to a (see Figure 5.4i);
- either a or b occur: mutual disabling conditions are use to define a choice between a and b, such that only one of the actions occurs. This choice defines that either the occurrence of b disables the occurrence of a, or vice versa. The textual notation for this behaviour is $\mathsf{B} = \{\neg a \rightarrow b, \neg b \rightarrow a\}$. The disabling condition $\neg b$ of action a is graphically represented by a solid arrow from b to a, with a vertical bar in between (see Figure 5.4ii);



(i) a follows b (ii) either a or b occur

(iii) a and b are synchronised (iv) independence of a and b

Figure 5.4: Some simple action relations

- a and b are synchronised: a synchronisation condition between a and b defines that both actions occur at the same time, or both actions do not occur at all. Synchronisation is a reciprocal condition for actions a and b, since a can only synchronise with b when b can synchronise with a. The textual notation for this behaviour is $\mathsf{B} = \{^{=}b \rightarrow a, {}^{=}a \rightarrow b\}$. The synchronisation condition $^{=}b$ of action a is graphically represented by a solid double lined from b to a (see Figure 5.4iii);
- independence of a and b: defines two initial actions a and b, with no dependency defined between them, therefore they may occur independently of each other at any time. In this case both actions have an associated start condition. The textual notation for this behaviour is $\mathsf{B} = \{\surd \rightarrow a, \surd \rightarrow b\}$. The start condition is graphically represented by a solid arrow pointing to the action (see Figure 5.4iv).

5.2.4 Conjunction of causality conditions

Often one needs to represent that multiple basic conditions involving different actions must all be satisfied to enable the occurrence of other action. Conjunctive causality conditions can be used to define the conjunction of two or more causality conditions involving different actions. The *and*-operator '$\wedge$' allows the representation of the conjunction of two causality conditions. The conjunction of multiple conditions can be represented by the repeated application of the same *and*-operator.

The conjunction of two causality conditions $\gamma 1$ and $\gamma 2$ of result action a is defined as follows: $\gamma 1 \wedge \gamma 2 \rightarrow a$. This definition indicates that $\gamma 1$ and $\gamma 2$ are both necessary conditions for the occurrence of result action a, Therefore action a is only allowed to occur when both conditions $\gamma 1$ and $\gamma 2$ are satisfied. The use of the *and*-operator is illustrated with the following examples:

- $a \wedge d \wedge {}^{\neg}b \rightarrow c$, specifies that action c is allowed to occur after actions a and d have occur and when action b has not occurred before c nor occurs simultaneously with c;

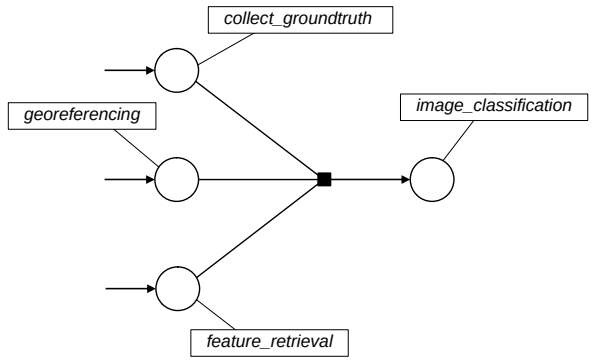


Figure 5.5: Conjunction of causality conditions

- $a \wedge \neg b \wedge =d \rightarrow c$, specifies that action c is allowed to occur simultaneously with d after action a has occurred and when action b has not occurred before c nor occurs simultaneously with c.

Figure 5.5 depicts an example of the conjunction of enabling conditions. In this figure the symbol '■' is used to graphically represent the conjunction operator. Action *image_classification*, r for short, models the classification of a satellite image. The actions *ground_truth*, *georeferencing* and *feature_retrieval*, $c1$, $c2$ and $c3$ for short, model activities that produce inputs required by action r of generate its results. The conjunction models that the classification of an image can only be finished successfully after all three other actions have finished. The textual representation of the behaviour depicted in Figure 5.5 is as follows:

$$\begin{aligned} B := \{\ & c1 \wedge c2 \wedge c3 \rightarrow r, \\ & \surd \rightarrow c1, \\ & \surd \rightarrow c2, \\ & \surd \rightarrow c3 \qquad \} \end{aligned}$$

5.2.5 Disjunction of causality conditions

When creating composite causality conditions, often one needs to represent that at least one simple or conjunctive causality condition must be satisfied for a certain action to happen. Disjunctive causality conditions define alternatives between two or more basic or conjunctive causality conditions. The or-operator '$\vee$' allows the representation of the disjunction of disjunctive causality conditions.

The disjunction of two causality conditions $\gamma 1$ and $\gamma 2$ of result action a is defined as follows: $\gamma 1 \vee \gamma 2 \rightarrow a$. This definition establishes that $\gamma 1$ and $\gamma 2$ are alternative causality conditions for the occurrence of the result action a, such that a is allowed to occur when at least one of these conditions is satisfied in an execution.

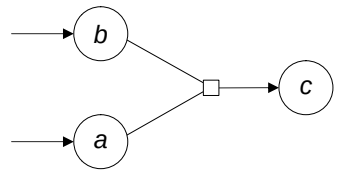


Figure 5.6: Disjunction of causality conditions

Figure 5.6 depicts behaviour $\mathsf{B} = \{a \vee b \rightarrow c, \surd \rightarrow a, \surd \rightarrow b\}$. In this Figure the symbol '□' is used to graphically represent the disjunction operator. In Figure 5.6 there are two possibilities: the occurrence of c either depends on the occurrence of a and is then independent of b, or the occurrence of c depends on the occurrence of b and is then independent of a.

This definition implicitly states that, according to what has been specified, an occurrence of c as a consequence of both the occurrence of a and the occurrence of b is not allowed. This is because the occurrence of c can not refer to the result of both occurrences a and b, when assuming that both a and b have occurred before c, because c it is only related to either a or b.

The abstract causality condition represented in Figure 5.6 describes, for example, the allocation of a particular resource, e.g., a stereoplotter. In such case, actions a and b account for two independent requests to use the stereoplotter, and action c represents the allocation of the stereoplotter to a particular task. In order to allocate the stereoplotter only one of the requests is necessary, since the resource can only be granted to one of them. The possible occurrence of another request is irrelevant to perform any further action.

Table 5.1 depicts a set of commonly used action relations between actions a, b, c and d. These relations represent definitions of consistent and comprehensible behaviours that consist of multiple actions composed as conjunctions and/or disjunctions of actions relations. The Table also shows the corresponding graphical notation for each behaviour.

5.2.6 Action attribute constraints

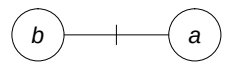
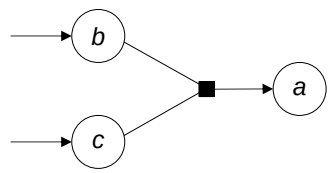
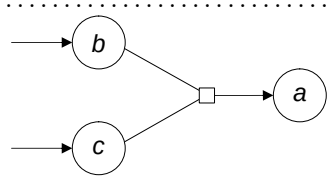
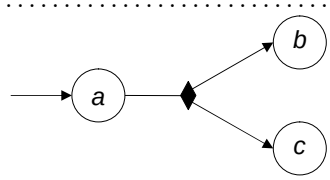
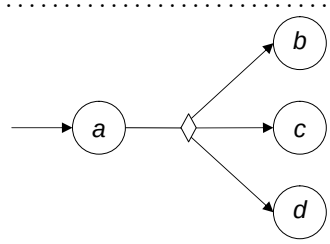
In section 5.2.1 we introduce the concept of the information, time and location attributes, which allow one to model the establishment of information, time and location values in action occurrences. In this section we extend this concept with *attribute constraints*, which allow to model the dependencies between information, time and location values established in different action occurrences. Action attribute constraints are part of the definition of causality relations.

The possible values that can be established in an attribute of a result action a are determined by three types of constraints:

- the *attribute value domain*, which defines the values that are allowed by result action a itself;
- *attribute reference relations*, which define how the attribute values of a depend on the attribute values of the actions in the causality condition of a;
- *attribute causality conditions*, which define how the occurrence of a depends on the attribute values of the actions in the causality condition of a.

This extension can be performed analogously to the information, time and location attributes of actions. Therefore we present attribute constraints in a uniform way for the three attributes.

Table 5.1: Common action relations

Action relation	Graphical Representation	Textual Representation
choice		$\{\neg a \rightarrow b, \neg b \rightarrow a\}$
and - join		$\{b \wedge c \rightarrow a,$ $\surd \rightarrow b,$ $\surd \rightarrow c\}$
or - join		$\{b \vee c \rightarrow a,$ $\surd \rightarrow b,$ $\surd \rightarrow c\}$
and - split		$\{a \rightarrow b, a \rightarrow c\}$
or - split (multiple choice)		$\{a \wedge \neg c \wedge \neg d \rightarrow b,$ $a \wedge \neg b \wedge \neg d \rightarrow c,$ $a \wedge \neg b \wedge \neg c \rightarrow d,$ $\surd \rightarrow a\}$

Attribute value domain

An action is used to represent a specific activity, for example, updating maps, measuring control points, classifying images, etc. This implies that an action only allows a certain range of attribute values to be established. The restriction of the universe of attribute values to a specific range is called the *attribute value domain* of a result action.

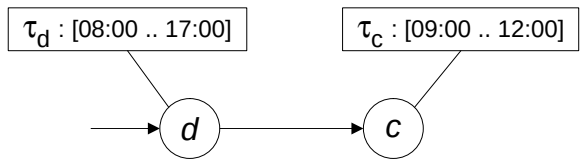


Figure 5.7: Attribute value domain

Figure 5.7 shows the definition of value domain for the information attribute of actions c and d. The time attributes are represented in separate text-boxes in the graphical notation, and are represented between brackets to the right side of the result action in the textual notation. In Figure 5.7 the symbols τ_c and τ_d represent the time values established in actions c and d, respectively. The range of allowed time attribute values of actions c and d is depicted between square brackets. The behaviour depicted in Figure 5.7 defines that c and d are only allowed to occur during the time ranges [08:00..17:00] and [09:00..12:00], respectively. The textual representation of this behaviour in as follows:

$$B := \{\ d \rightarrow c\ (\tau_c : [09{:}00..12{:}00]),\\ \surd \rightarrow d\ (\tau_d : [08{:}00..17{:}00])\ \}$$

Location and information values can be dealt with in the same way as the time values presented in Figure 5.7. For a given action, the set of values that are allowed as a result of the action is called the *information value domain*, the set of locations where the action occurrence make its results available is called the *location value domain*, and the set of the time moments when the action can occur is called *time value domain*. The definition of attributes value domains is optional, therefore it is only used when it is relevant at the considered abstraction level, otherwise it can be omitted.

Attribute reference relation

The attribute value established by some action occurrence may depend on the attribute values established in other action occurrences. In this case we say that an action occurrence *refers* to the attribute values of other action occurrences.

An attribute value reference from one action occurrence to another implies a dependency (relation) between these action occurrences. Therefore, an action c may refer

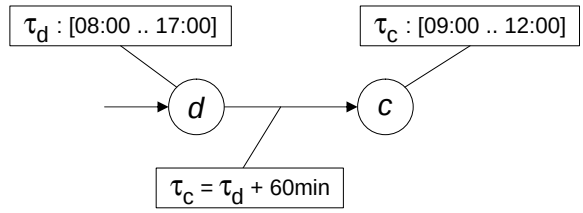


Figure 5.8: Attribute reference relation

to the attribute value established by another action d, when c depends on d, and c occurs after d.

Figure 5.8 depicts a behaviour in which action c refers to the time value of action d. The time value of action c depends on the time value established in action d such that the value of τ_c must be greater than the value of τ_d (which is implicit in the enabling relation) and less or equal to the value of τ_d plus 60 minutes. The relation between τ_c and τ_b is called a *time reference relation*. The conjunction of this time reference relation with the time value domains of c and d allows action c to occur within an hour after the occurrence of action d. That means action c can only occur if action d occurs between 08:00 and 11:00. The occurrence of action d at any other time will not enable the occurrence of action c, because the reference relation $\tau_d < \tau_c \leq \tau_d + 60\text{min}$ can not be satisfied. The textual representation of the behaviour in Figure 5.8 is as follows:

$$B := \{\ d \rightarrow c\ (\tau_c : [09{:}00..12{:}00])\,[\tau_c \leq \tau_d + 60\text{min}],\\ \surd \rightarrow d\ (\tau_d : [08{:}00..17{:}00])\qquad\qquad\qquad\qquad \}$$

The reference relation $\tau_c \leq \tau_d + 60\text{min}$ is graphically represented within a text-box linked to the enabling relation between the involved actions, in this case c and d. In the textual notation, information reference relations are represented as constraints between square brackets to the right side of the corresponding attribute of the result action.

Implicit time references

Some causality relations prescribe a specific time relation. For example, an enabling relation contains an implicit time constraint, which defines that the enabling actions must occur before the result action. That means, for instance in the relation $b \rightarrow a$ there is an implicit time constraint $\tau_a > \tau_b$ prescribed by the enabling relation between b and a. This time constraint is called an *implicit time reference relation*. Implicit time reference relations must be made explicit whenever we have to determine the complete time attribute constraint of some result action to avoid impossible behaviours.

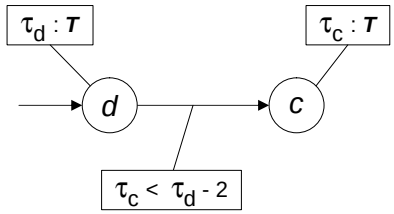


Figure 5.9: Implicit time reference (impossible action)

For example, Figure 5.9 defines the implicit time reference relation $\tau_c > \tau_d$ prescribed by the enabling relation between actions *b* and *a*. The time reference relation $\tau_c < \tau_d - 2$ implies that τ_c must be smaller than τ_d. In other words, action *c* may occur before *d* occurs. This creates a conflict with the implicit time constraint $\tau_c > \tau_d$ prescribed by the enabling relation between *d* and *c*, which means that action *c* will never occur in this behaviour definition.

The sane consideration have to be taken into account when defining time relations between synchronised actions, in which the synchronisation relation defines an implicit time reference relation that imposes that all synchronised actions must occur at the same time moment.

Attribute causality condition

A causality condition can be extended to define conditions on the attribute values of the enabling actions. These conditions are called *attribute causality conditions*, and must be satisfied in order to allow the occurrence of the result action.

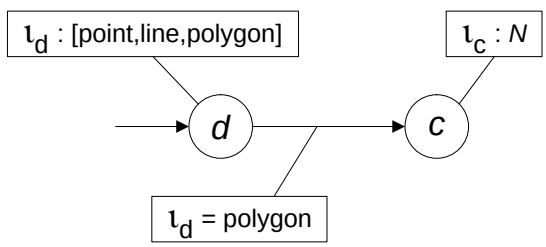


Figure 5.10: Attribute causality condition

Figure 5.10 depicts an example of an information causality condition. Action *d* represents the extraction of a geometric feature from a data collection, which has geometry type point, line or polygon. The information value domain of action *d* is defined as ι_d: [point, line, polygon]. Action *c* represents the calculation of the area of features. The information value domain of action *d* is defined with the symbol '*N*' which represents any natural number. The information causality condition ι_d = polygon represents that

action c is only allowed to occur when the information value established in d is equal to polygon. The textual representation of the behaviour in Figure 5.10 is as follows:

$$B := \{\ d\ [\iota_d = \mathsf{polygon}] \rightarrow c\ (\iota_c : \mathrm{N}),\\ \surd \rightarrow d\ (\iota_d : [\mathsf{point},\mathsf{line},\mathsf{polygon}])\ \}$$

Attribute causality conditions are represented within a text-box linked to the enabling relation between the actions in the graphical notation. In the textual notation, information causality conditions are represented as constraints between square brackets to the right side of the corresponding causality condition.

5.3 Decomposition

We have introduced so far a set of comprehensive design concepts suitable to specify systems at a single level of abstraction. Decomposition is therefore needed to obtain more detailed system descriptions, i.e., specifications at a lower level of abstraction. For this purpose we introduce entity decomposition and action decomposition.

5.3.1 Entity decomposition

An entity may consist of other, lower-level, entities or sub-entities. In order to add internal structure, an entity can be decomposed into two or more sub-entities. These sub-entities should be interconnected via internal interaction points, to be able to cooperate. Furthermore, the interaction points of the original entity should be maintained.

To be able to represent an entity in this way, we need to know the internal structure of the system that is represented by the original entity, in terms of the parts that compose the system. Figure 5.11 depicts a decomposition of the GI-system entity of Figure 5.1, into five sub-entities, which are interconnected via five internal interaction points.

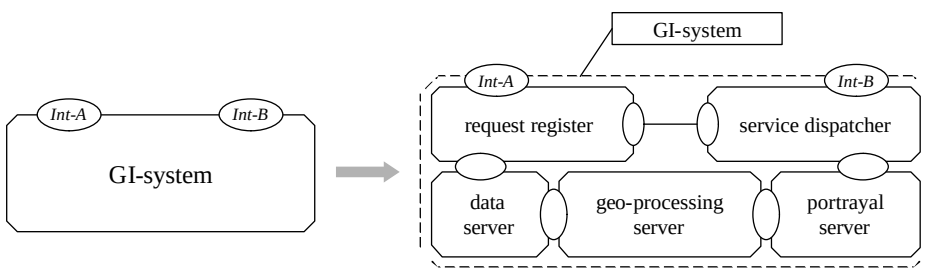


Figure 5.11: Entity decomposition (refinement)

The sub-entities of Figure 5.11 represent the sub-systems that define the internal structure of the GI-system entity. Since the entity structure of the decomposition adds more detail to the initial representation, this new entity structure is called a refinement of the GI-system entity. To reflect this the GI-system entity is represented in Figure 5.11 as a dashed polygon containing the decomposition. The inverse of decomposition is composition, which allows one to abstract from (parts of) the internal structure.

5.3.2 Action decomposition

In order to obtain a more detailed model of an activity, this activity has to be decomposed into multiple sub-activities and their relations. The relevant characteristics of these sub-activities can be modelled again by distinct actions at a lower abstraction level. The concepts necessary to model relations between actions are explained in detail in section 5.2.3.

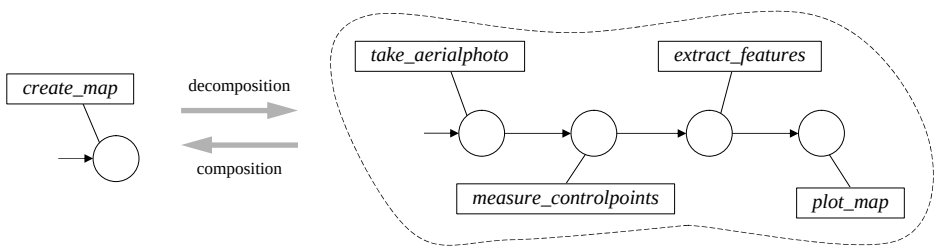


Figure 5.12: Action decomposition

Figure 5.12 depicts the modelling of the activity of creating a map, at two different abstraction levels. At the most abstract level, a single action *create_map* models what result is established by the entire activity Create map. At a more detailed level, the activity Create map is decomposed into four related sub-activities, which are modelled by four distinct actions.

The arrows connecting the actions depicted in Figure 5.12 represent enabling relations between the actions. For example, action *extract_features* may occur only after action *measure_controlpoints* has occurred. Intuitively, these models are considered consistent if the result of action *plot_map*, which is the final action of activity Create map, corresponds to (conforms to) the result of action *create_map*.

An action may also model an integrated interaction, which abstracts from the individual interaction contributions of the involved entities. Replacing an action by an interaction is considered a refinement of the action, since an interaction defines a specific distribution of the responsibility for performing the interaction over multiple entities. We call the replacement of an action by an interaction *action distribution*. All constraints on the interaction are combined into the corresponding action. Figure 5.13 depicts the distribution of action measure that is refined into the interaction

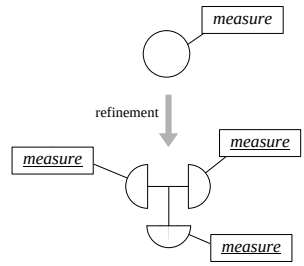


Figure 5.13: Action distribution

measure consisting of three interaction contributions.

5.4 Behaviour Structuring

In this section we discuss the techniques to structure behaviours as compositions of smaller or simpler sub-behaviours. A structured behaviour is in general easier to read and understand than an unstructured (monolithic) behaviour. Structuring large behaviours also enables the identification of sub-behaviours that could happen multiple times, and in these cases one can create a structured behaviour that defines a behaviour in terms of a repetition of a (simpler) behaviour (see Figure 5.18). An important aspect of behaviour structuring is that it allows one to decompose a complex behaviour into sub-behaviours that can be assigned to different entities.

We distinguish This chapter two techniques to structure behaviours: causality-oriented behaviour structuring and constraint-oriented behaviour structuring. These techniques can be used separately and in combination.

5.4.1 Causality-oriented structuring

In the structuring of a complex behaviour, one may want to define this behaviour in terms of sub-behaviours and the relations between these sub-behaviours. Relations between (sub-)behaviours play the same role in behaviour definitions as relations between actions, except at a different level of granularity: behaviours can be defined as compositions of actions, as compositions of sub-behaviours, or a combination of both. The causality-oriented structuring technique allows the structuring of a complex behaviour in terms of less complex sub-behaviours and their relationships. That means in a causality-oriented structured behaviour, conditions in one behaviour definition enable or disable actions in another behaviour.

A relation between two or more behaviours can be defined by introducing entry and exit points. Entry points and exit points are only syntactical constructs that enable the decomposition of causality relations. This allows for an action and the condition

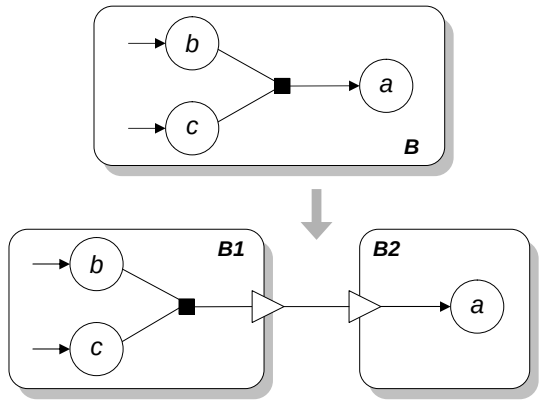


Figure 5.14: Causality-oriented structuring

for its occurrence to be defined in distinct sub-behaviours. Therefore, entry and exit points of monolithic behaviours are applied in the following way: an *exit point* in some behaviour defines a causality condition that can be used to enable actions in other behaviours; an *entry point* in some behaviour allows actions in this behaviour to be enable by conditions defined in other behaviours. Consequently, entry points and exit points have to be coupled together to define a causality relation where actions of a latter behaviour become dependent of actions of a former behaviour.

Figure 5.14 illustrates the decomposition of causality relation $b \wedge c \rightarrow a$ into the (pseudo)causality relations $b \wedge c \rightarrow \mathsf{exit}$ and $\mathsf{entry} \rightarrow a$, which are defined in behaviours *B1* and *B2*, respectively. Entry points and exit points are graphically represented by the symbol '$\triangleright$', pointing inside and outside the corresponding behaviour, respectively. Causality relation $b \wedge c \rightarrow \mathsf{exit}$ defines that the exit point of *B1* represents causality condition $b \wedge c$. Causality relation $\mathsf{entry} \rightarrow a$ defines an entry point in the behaviour *B2*, representing a place-holder for the causality condition of a.

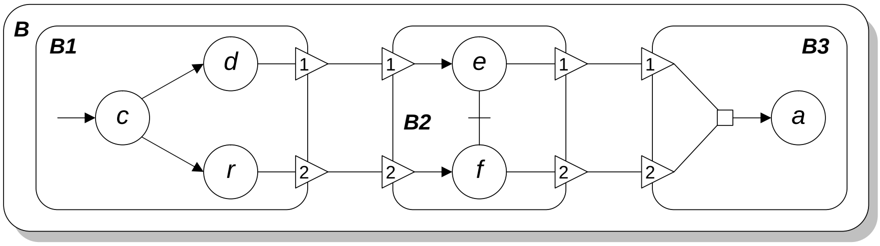
The textual representation of the behaviours depicted in Figure 5.14 are as follows:

$$B := \{\ b \wedge c \rightarrow a,\ \surd \rightarrow b,\ \surd \rightarrow c\ \}$$

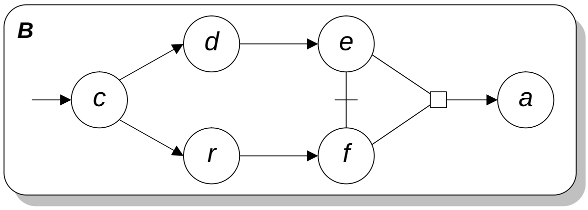
$$B1 := \{\ b \wedge c \rightarrow \mathsf{exit},\ \surd \rightarrow b,\ \surd \rightarrow c\ \}$$

$$B2 := \{\ \mathsf{entry} \rightarrow a\ \}$$

By combining the exit point of *B1* with the entry point of *B2*, condition $b \wedge c$ is assigned as the causality condition of action a. This corresponds to replacing entry in the causality condition of a by causality condition $b \wedge c$. The combination of an exit and an entry point is graphically represented by linking the corresponding exit and entry symbols with a solid line. The causality-oriented structured causality relation represents the same condition as the unstructured causality relation.



(i) Causality-oriented behaviour composition structure



(ii) Monolithic behaviour structure

Figure 5.15: Multiple entry points and exit points

Multiple entry and exit points

Figure 5.15 illustrates the use of entries and exits to compose behaviours from sub-behaviours. Figure 5.15(i) depicts the composition of behaviour B from sub-behaviours $B1$, $B2$ and $B3$. Figure 5.15(ii) depicts the corresponding monolithic definition of B.

The sub-behaviours in Figure 5.15(i) have multiple entries or exits. In order to distinguish between multiple entries and exits of a single sub-behaviour, the keywords exit and entry are appended with a unique identifier. In the graphical representation, this identifier is depicted inside the symbol '$\triangleright$'. Here, we have used natural numbers as identifiers for entries and exits in a behaviour. The textual representation of behaviour B as depicted in Figure 5.15(i) is as follows:

$$
\begin{aligned}
B = \{\, & B1.\mathsf{exit1} \rightarrow B2.\mathsf{entry1},\ B1.\mathsf{exit2} \rightarrow B2.\mathsf{entry2},\\
& B2.\mathsf{exit1} \rightarrow B3.\mathsf{entry1},\ B2.\mathsf{exit2} \rightarrow B3.\mathsf{entry2}\\
& \text{where}\\
& B1 = \{\, c \rightarrow d \wedge r,\ d \rightarrow \mathsf{exit1},\ r \rightarrow \mathsf{exit2} \,\},\\
& B2 = \{\, \mathsf{entry1} \wedge \neg f \rightarrow e,\ \mathsf{entry2} \wedge \neg e \rightarrow f,\ e \rightarrow \mathsf{exit1},\ f \rightarrow \mathsf{exit2} \,\},\\
& B3 = \{\, \mathsf{entry1} \vee \mathsf{entry2} \rightarrow a \,\} \qquad \}
\end{aligned}
$$

Parameterised exits and entries

The association of a causality condition with an exit or entry point is called a (pseudo-)causality relation. However, in contrast to a result action, action attributes

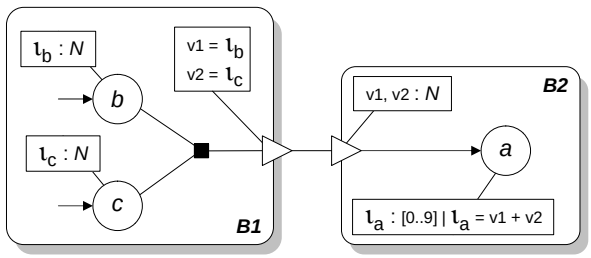


Figure 5.16: Parameterised exits and entries

can not be associated with an exit or entry point, since exit points and entry points do not model activities. Instead, exit and entry points are introduced as syntactic constructs that allow one to define a result action and (parts of) its causality condition in different sub-behaviours.

For the situations where actions in one behaviour need to refer to the result values of actions in another behaviour, we allow that an entry or an exit can be parameterised with a list of information, time or location variables. These variables should hold all information passed from the exit point to the entry point. These variables contain a selection of the information, time and location values of the enabling actions that can be referred to when the causality condition associated with this entry or exit is satisfied. A requirement on the combination of an exit and an entry is that both have the same parameter list, i.e., the same number of variables, having the same type and specified in the same order. Only the variable names of these parameter lists may differ.

Consider the example of Figure 5.16. Actions b and c establish information values, and action a has to refer to the information values of b and c. For this purpose, the exit point of $B1$ and the entry point of $B2$ are extended with a parameter list consisting of two information variables v1 and v2. The resulting causality relations obtained with this extension are defined as follows:

$$
\begin{aligned}
B = \{\, & \surd \rightarrow B1.\mathsf{entry1},\ \surd \rightarrow B1.\mathsf{entry2},\ B1.\mathsf{exit} \rightarrow B2.\mathsf{entry} \\
& \text{where} \\
& B1 = \{\, \ldots,\ b\,(\iota_b : N) \wedge c\,(\iota_c : N) \rightarrow \mathsf{exit}\,(\mathsf{v1},\mathsf{v2} : N\,[\mathsf{v1} = \iota_b,\ \mathsf{v2} = \iota_c]\,\}, \\
& B2 = \{\, \mathsf{entry}\,(\mathsf{v1},\mathsf{v2} : N) \rightarrow a\,(\iota_a : N)\,[\iota_a = \mathsf{v1} + \mathsf{v2}]\,\} \qquad\qquad \}
\end{aligned}
$$

The statement $B1.\mathsf{exit} \rightarrow B2.\mathsf{entry}$ is allowed, since the parameter lists of $B1.\mathsf{exit}$ and $B2.\mathsf{entry}$ match. This statement implicitly defines that the parameters of $B2.\mathsf{entry}$ get the same values as the corresponding parameters of $B1.\mathsf{exit}$.

Behaviour recursion

Some behaviours in real life consist of the repeated execution of a sub-behaviour. For example, the behaviour of a quality control process can be considered as the repetition of the behaviour revise quality of product, in which an instance of this behaviour is created each time the product is rejected. Figure 5.17 depicts the behaviour of a product delivery chain, in which action c models the checking of a product, action r models the revising and improving of a rejected product, action p models the packing of the product and action s models the sending of the product.

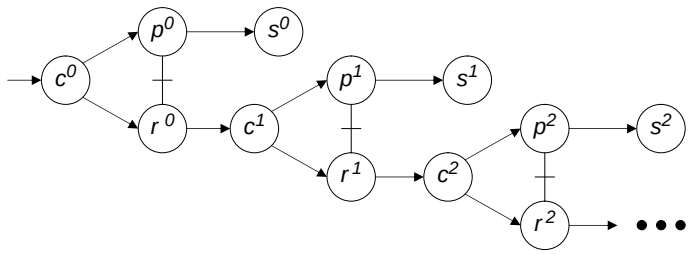


Figure 5.17: Repetition of a sub-behaviour

The same repetitive behaviours can be modeled using the recursive instantiation of a behaviour. Figure 5.17 shows the graphical representation of behaviour B, which models the behaviour of a product delivery chain by the recursive instantiation of sub-behaviour D, which models the quality check and delivery of a product.

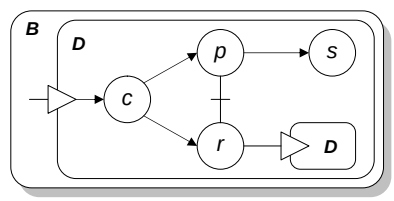


Figure 5.18: Recursive instantiation of sub-behaviour D

The textual notation of the recursive instantiation of sub-behaviour D of behaviour B is as follows:

$$
\begin{aligned}
B = \{\, & \surd \rightarrow D.\mathsf{entry} \\
& \text{where} \\
& D = \{\, \mathsf{entry} \rightarrow c, \\
& \qquad c \rightarrow p \vee r, \\
& \qquad p \rightarrow s, \\
& \qquad r \rightarrow D.\mathsf{entry} \,\} \,\}
\end{aligned}
$$

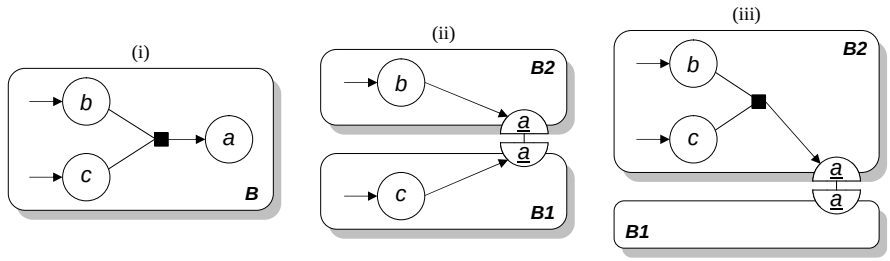


Figure 5.19: Constraint-oriented decomposition

5.4.2 Constraint-oriented structuring

An action can be defined in distributed form as a composition of interaction contributions. Following the same reasoning, a behaviour can be defined in distributed form as a composition of interacting sub-behaviours. The structuring of a behaviour in terms of a composition of interacting sub-behaviours is called *constraint-oriented structuring*. The constraint-oriented structuring technique allows one to decompose complex conditions and constraints on the execution of an action into simpler sub-conditions and sub-constraints that are assigned to interaction contributions defined in separate sub-behaviours. Furthermore, the constraint-oriented structuring technique is needed to structure a behaviour in sub-behaviours, such that each sub-behaviour can be assigned to an entity in an entity structure.

Figure 5.19 depicts the decomposition of action *a* into two interaction contributions that are assigned to behaviours *B1* and *B2*. In the case of Figure 5.19(ii) action *a* has been distributed such that the interaction contribution $\underline{a}$ of *B1* depends on the occurrence of *b* and the interaction contribution $\underline{a}$ of *B2* depends on the occurrence of *c*. Since an interaction can only occur when both interaction contributions can occur, interaction *a* can only occur after both *b* and *c* have occurred. Consequently, the condition for the occurrence of action *a* is exactly the same as the condition for the occurrence of interaction *a*. Figure 5.19(iii) depicts a different assignment of actions *b* and *c* to sub-behaviours *B1* and *B2*, namely that actions *b* and *c* are both assigned to *B2*.

Chapter 6

The external perspective

A child on a farm sees a plane fly overhead and dreams of a faraway place.
A traveller on the plane sees the farmhouse and dreams of home.

Carl Burns

This chapter introduces the so called 'external perspective model', which is used to define a GSI system's external behaviour by explicitly identifying the boundary level interactions in which the system and its environment participate.

The external perspective model is useful to provide a specific description of individual services from the user's point of view. It provides insight in the relations between the system and its environment and it also serves as the basis for developing internal perspective models in subsequent design phases.

The structure of the chapter is organised as follows: section 6.1 explains basic aspects of GSI systems and their working environments; section 6.2 describes the trajectory to obtain a external perspective design; section 6.3 provides the necessary concepts to create external perspective models; section 6.4 introduces some specific data types that are needed to handle geographic data; and section 6.5 describes the techniques to create external perspective models according to our design trajectory and illustrate it with the simple example of a service definition.

6.1 The GSI system

Figure 6.1 shows a GSI system composed of a set of Geo-Service Providers (GSP). Each provider focuses on certain needs of one or more user communities. A user community [OGC03b] is collection of people (a government agency or group of agencies, a

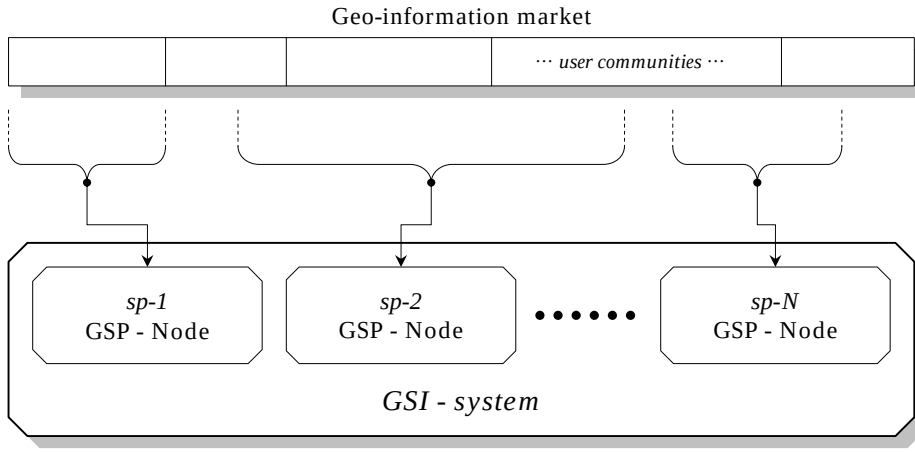


Figure 6.1: The GSI as a target oriented system

profession, a group of researchers in the same discipline, corporate partners cooperating in a project, etc.), who share a common digital geographic information language and common spatial feature definitions, at least part of the time. This implies a common view of the world as well as common abstractions, feature representations, and metadata.

Each provider has accordingly different responsibilities towards the community and therefore its own set of requirements. Service providers may range from just data producers, collecting foundation data, to specialised information developers that exploit data to generate tailored products to support different applications (planning, decision making, military operations, etc.) in the market.

GSI systems are therefore complex systems that encompasses multiple independent entities. These entities benefit from each other's functionality and together realise the services of the system. Services are seen as the result of the combined effect of the activities within the system on its surrounding environment. This means that one or more of the system's entities work in a coordinated manner to provide services.

To make sense of these systems, however, and for someone to properly and systematically develop them, it is necessary to select a specific aspect of the system to be used as the basis for development. Since the overall functionality of GSI systems varies as explained above, we do not start capturing or gathering the overall system requirements, but we rather focus on the organisation and interpretation of gathered requirements (according to a provider) to form services. This is done by means of external perspective models.

External perspective models (EP models) also called services models, represent simplified views of the system that are used to define a service that complies to a given set of functional requirements. Each EP model describes only a subset of the overall

functionality of the system, and therefore represents a partial system specification.

6.2 Design trajectory

During the service design phase of the GSDM methodology we concentrate on the system according to the point of view of the users. We use successive design steps in a top-down design trajectory to develop external perspective (EP) models. The developed models are stored in the repository where they can be instantiated, used as reference for developing detail specifications (internal perspective models) or used as a building block in more complex service realisations .

We work out this development phase in two major development steps: the service definition and the extended service definition (see Figure 6.2).

During the *service definition* development step we create a so called observable behaviour of the system. This behaviour describes the system from the point of view of the user. This is done by creating an entity model that discriminates or makes a clear distinction between the system and its environment. Such model defines the roles of the *service user entity(ies)*, the *service provider entity*, and the interconnection structured between the identified entities.

In certain cases the generation of a service requires the use of functions that are not the concern of any GSI provider. Examples include, a.o., bank transactions and postal

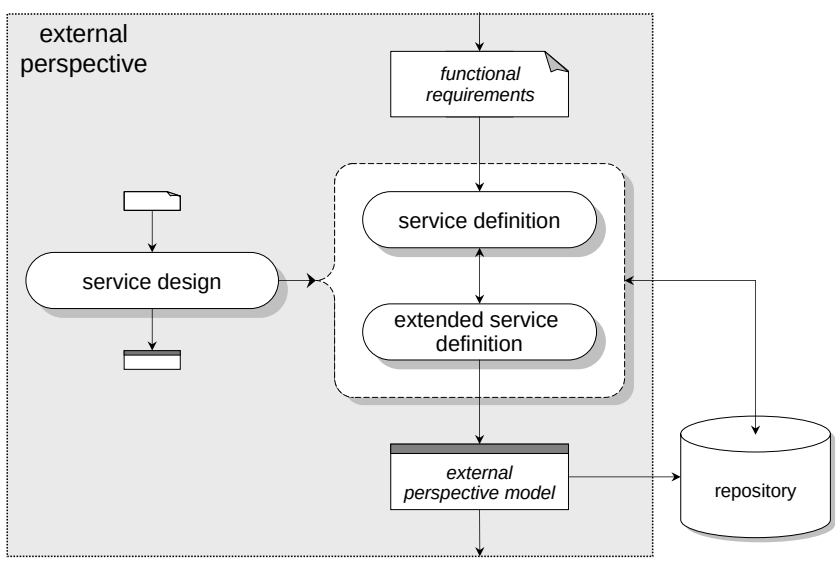


Figure 6.2: Design trajectory at the external perspective level

delivery. The *extended service definition* development step is used for such cases. The purpose of this step is to enhance the existing service definition in order to describe the existence of entities responsible for functions that are not native to the GSI system. These functions do not form part of the GSI core, and thus these functions are considered foreign functionality. The entities that provide these functions are called *auxiliary entities*.

The extended service definition is therefore a refinement of the service definition in which the service provider entity is decomposed to show the existence of auxiliary entities in a service realisation. In those cases where such foreign functions are not required then extended service definitions are not necessary.

Once the relevant entities have been identified, we proceed to fully specify the shared boundary between these entities. This specification comprises the interactions in which the service user entity and the service provider entity participate, the relations and ordering of these interactions, and the definitions of the items manipulated or exchanged in these interactions.

6.3 Design concepts

This section introduces a specialisation of some of the architectural concepts introduced in Chapter 5. These concepts are used in the representation of systems according to the external perspective level.

6.3.1 Functional entity

We use an entity as an abstract concept to represent (a part of) the system, or (a part of) the system's surrounding environment capable of interacting with the system. An entity does not exist in isolation, but rather it is embedded in an environment that consist of other entities. Therefore, an entity is important for what it can do for other entities, for what functionality it can provide to its environment. The entity concept abstracts from the characteristics of the system or system part that it represents.

Within a service definition we distinguish three different types of entities or entity roles: the service provider entity, the service user entity, and the auxiliary entity.

From the point of view of the environment the entity that provides certain functionality is known as the *service provider entity*. The environment of the service provider entity consists of other entities. These environment entities are either client entities or auxiliary entities. An environment entity that is capable of interacting with the system and uses its functionality is called a *service user entity*. Client entities consist of people or other systems that use the functionality provided by the service entity.

For the particular cases where foreign functions may be required by the system in the

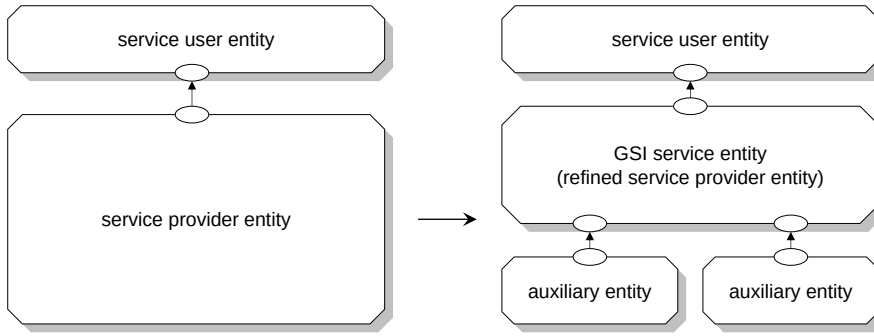


Figure 6.3: Different types of functional entities

realisation of a service (say from other application domains), the service entity may interact with other environment entities. These entities are called *auxiliary entities*, they consist of other systems, e.g., a banking system, a speech recognition system, etc. Auxiliary entities in principle do no use the main functionality provided by the service provider entity, but they rather support the service provider entity with the provision of its functionality [Far02].

For any entity to use the functionality provided by another entity, both entities have to interact with each other. Therefore entities are equipped with interaction points indicating the mechanism through which entities can interact. It is only through its interaction points that the functionality of an entity can be accessed. For two entities to interact they must form an interaction point relation.

Figure 6.3 depicts the different entity types and their interaction structure. Entities are shown as non-overlapping rectangles with cut-off corners whereas interaction points are depicted with ovals that overlap with the entity and are connected by a line.

6.3.2 Interactions

During the creation of (extended) service definitions we describe how entities cooperate to realise a service. This description represents collaboration between these entities. The definition of collaboration between two entities, say a service user entity and the service provider entity, has the following implications:

1. The two entities are interconnected in a defined way so that they can exchange information;

2. The two entities can affect each others functional behaviour in a defined way through the exchange of information;

These facts are formalised using the notions of interactions and interaction attributes. For our specific goals we define a partition of the information attribute. The information attribute of an interaction models the establishment of these information values. What information values are established depends on the contribution of each behaviour on the interaction. This information values can take three forms: parameters, inputs or results. Parameters, inputs and results are a specialisation of the information attribute (see Section 5.2.1). Parameters, inputs and results basically define a partition of the possible types of values that can be established in an interaction.

Parameters are used to configure the behaviour that receives information in the interaction. These parameters contain attribute values that are used to evaluate conditions (causality conditions) within the behaviour. For example, if there exists a behaviour definition to, say, determine an optimum route between two points, and this definition includes an alternative condition to obtain routes of different nature or quality (e.g. shortest, quickest, cheapest), then, parameters are used to specify, for instance, whether to calculate the *shortest* route or the *cheapest* route between the points. Parameters are considered optional information values and for this reason they are always specified with a default value.

Inputs are used to define the information that is required to enable the instantiation of a behaviour and that therefore must be provided at an interaction point. Considering the previously mentioned example, the 'optimum route' service requires information about the *starting* and *ending* points of the route to be able to realise any calculation. This information is required for the interaction to take place. In the absence of any of these inputs this specific service can not be used. A combination of the set of parameters and inputs would be required in the case that the 'optimum route' behaviour definition could take into consideration, e.g., intermediate points.

Results are used to describe the outcome of a service, which is made available at an interaction. In the case of the route determination the results could encompass a raster image showing some background detail (roads, buildings, etc.) and a vector

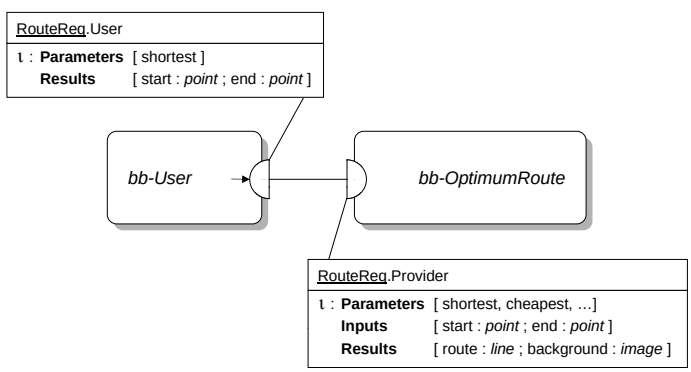


Figure 6.4: Special case of the information attribute of an interaction

line showing the route connecting the given points. It could also be just a vector line indicating the route. Alternative results can be configured by the definition of parameters.

Figure 6.4 shows an simple representation of the optimum route behaviour definition. The three types of information values are only specified if applicable; some services, for example, may not require parameters.

6.4 Spatial data types

During the development of a service definition, we fundamentally create a composition of architectural elements and we put strong emphasis in the definitions of the relationships between the elements. We use the concept of interaction to model these relationships. Each element participating in an interaction imposes constraints that define under which conditions it can participate in the interaction and which are the results that can be established as the outcome of the interaction.

In the case of a processing element, for example, these constraints can define the requirements, in terms of data types, that would allow the service provided by this element to be used. Therefore interaction constraints depend highly on the use of data types. Data types are also indispensable for the characterisation of the results of any given processing operation.

To facilitate the organisation of interaction constraints we have structured the information attribute in terms of parameters, inputs and results (see section 6.3.2). However, since these interactions constraints depend on the data types, we introduce a set of data types specifically tailored for the manipulation of geographic data.

The ISDL metamodel [Qua03] provides two kinds of data types, primitive data types and defined data types (see Figure 6.5). In addition to the conventional *primitive data types* (integer, boolean, string, etc.), we introduce the *defined data types* geo-object, theme, composite and collection. We call these spatial data types [RSV01]. Figure 6.5 depicts the data types hierarchy according to the needs of GSDM.

Figure 6.5 shows the relation between DataType with two data types PrimitiveDataType and DefinedDataType. Primitive data types include a.o., integer, string, boolean (not shown in Figure 6.5). Other data types, such as spatial data types, are sub-groups of the DefinedDataType. The type geoFeature, depicted in Figure 6.5, represents the data types that can be used to define geographic data.

The geographic object type (geoObject) is used to represent single entities in the real world. A geographic object has an associated set of attributes that provide semantic description to the object. For example, a geographic object to represent a *city* may have attributes such as name, population and foundation date.

A geographic object has also an associated spatial description, which is represented

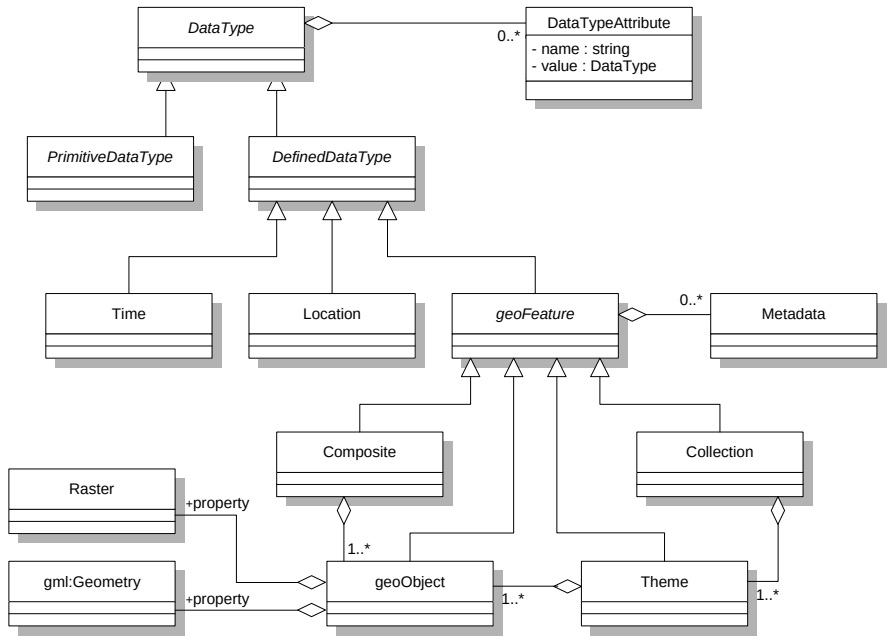


Figure 6.5: The data type concept

by its position, size, shape, orientation and reference system. The spatial description of a geographic object is defined in its gml:Geometry property. The geometry of a geographic object is defined using the Geographic Markup Language (GML) base schema [OGC03a] for the representation of feature geometries (see Appendix C for an overview of the GML feature schema).

When the spatial description of a geographic object is provided based on imagery (grided data), then the spatial description is defined in the raster property of the object. This applies, for example, to land use or soil type objects.

The theme type (Theme) is used to represent a group of homogeneous geographic objects that share the same structure, in terms of the set of attributes and geometry. A theme can be used to organise a group of cities, all of which exhibit the same geometry and have consistent values for the associated attributes.

A domain is used to delimit the scope of the theme. A domain, which can be geometric or thematic, acts as a constraint and specifies which objects are allowed as members of the theme. For example, objects of a theme have properties that distinguish them from objects that do not belong to this theme. For instance, one can define a city theme with the thematic domain "more than 100000 inhabitants", which means that only cities that are large enough to satisfy this constraint can become members of this city theme. The domain can also be geometric, such as a minimum size (area) or a predefined area of interest, for instance, within a radius of 100 km of point X.

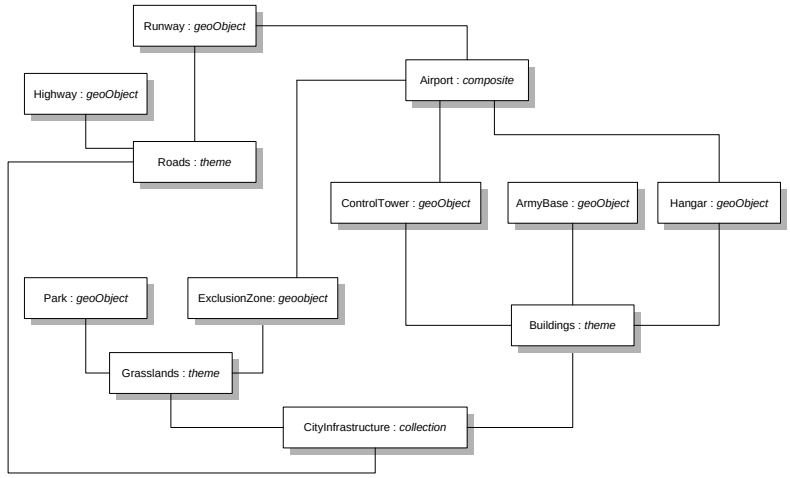


Figure 6.6: The use of spatial data types

Themes can be grouped in collections, and for this purpose the collection type (Collection) has been introduced. A collection can be used to represent a complete data set, which contains a series themes, e.g., rivers, buildings, roads, etc.

The composite type (Composite) is used for the representation of groups of heterogeneous geographic objects, that is, objects with different geometries and semantic descriptions, but that have a specific relationships with each other. These relationships are application-oriented and are determined by the views of the users on the data. Figure 6.6 a model created using the newly introduced spatial data types. The Figure shows an *airport* as a complex object that is composed of other geographic objects such as *runways (roads)*, *hangers (buildings)*, *exclusion zones (grasslands)*, etc. This type of objects can be represented using the composite type. All objects within a composite keep their individual structure, but together they represent an specific real world entity of interest in a particular application domain, such as, an airport.

Whenever necessary, more complex data types can be defined making use of the abstract class DefinedDataType. The definition of a new data type consists of the data type name, and zero, one or more attributes (class Attribute), where each attribute has a name and a value of certain a type.

6.5 Service design

The design trajectory of the external perspective defines that the specification of GSI services is done in two steps: the service definition and the extended service

specification. Here we describe the techniques to create these external perspective models.

6.5.1 Service definition

As specified in the design trajectory, during the service definition phase the first task is to identify the various entities involved in the service, namely, user entity, service entity and auxiliary entities if applicable. This separates the system from its environment and more importantly helps identifying the users of the service.

The TD-service is used as a running example to illustrate the different steps taken during the service design phase. More detail on the characteristics of the TD-service is given as it becomes necessary.

Figure 6.7 shows a simple example of a service model. The figure depicts the tax determination service, TD-service for short. The TD-service deals with: registration of tax payers for taxation, which takes place when a person acquires taxable land; the determination of taxes on exploited land; and the delivery of invoices to users on periodical basis.

For the case of the TD-service, we have identified the so-called *Taxation module* as the service provider entity and the so-called *Tax-payer* as the service user entity (see Figure 6.7). The service user entity in this case is depicted in Figure 6.7 with a double line to represent that multiple tax payers may benefit of the functionality provided by the service provider entity. Figure 6.7 also shows an interaction point as the mechanism that establishes the interconnection structure between the service entity and the client entities.

After defining the user and service entities, the next task is to specify the interactions

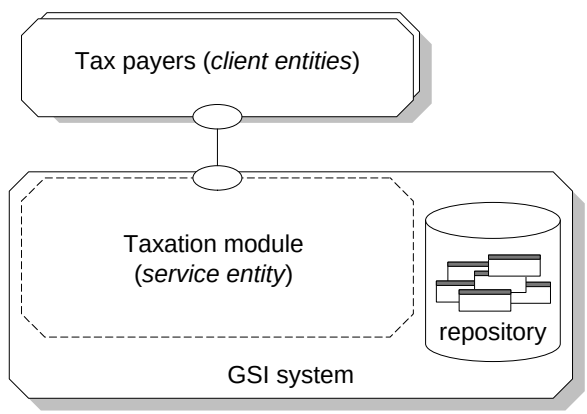


Figure 6.7: TD-service definition

in which these entities are involved and the relationships between these interactions. This is valid for the interactions between the service entity and the user entity(ies), as well as for the interactions between the service entity and the auxiliary entities. Therefore we explain this step in section 6.5.2 together with the description of the extended service definition step.

6.5.2 Extended service definition

There are two purposes for the extended service definition : 1. to identify all auxiliary entities that support the service entity for the realisation of the service; and 2. to determine all the interactions between the the service entity and the auxiliary entities. This step is only necessary if the participation of auxiliary entities is necessary in a service realisation.

The difference between developing a GSI-service definition and its extended definition is that in the former case we abstract from the interactions between the system and the auxiliary entities and in the latter case we do not.

Consider, for example, the sequence of interactions as depicted in Figure 6.8right, in which the occurrence of interaction *Int-A'* is followed by the occurrence of *Int-B*, which is followed by the occurrence of *Int-C*, which is followed by the occurrence of *Int-D'*. This would correspond to the *extended service definition*. However, to specify the *service definition* we would only include interactions *Int-A* and *Int-D* and the relationship between their occurrences in which the occurrence of *Int-A* is followed by the occurrence of *Int-D*.

Therefore, the techniques used to define interactions according to both levels are the same. The only difference resides on the type of entities being considered. Based on this consideration, we discuss only how to identify and specify interaction patterns independent of whether the interactions are used on a service definition or on a extended service definition.

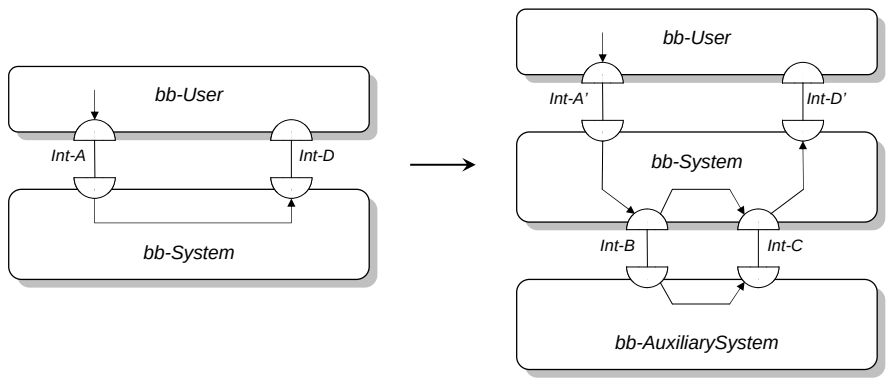


Figure 6.8: Interactions at the service and extended service levels

In order to specify the interactions of a service, we have to associate a behaviour block to each identified entity. By the end of the extended service definition phase, all participant entities should have a behaviour definition associated to them. However, at this level of development, these behaviour specifications only represent external behaviour, which fundamentally describe all border level interactions of the service entity, and the relationships between those interactions. The relationships between interactions basically capture the order in which interactions can take place and their information dependencies.

A extended service definition describes both the interactions between the service entity and its client entities; and the interactions between the service entity and any auxiliary entities, together with their ordering. This description takes into account the external functions that may be used by the service entity in order to provide its own service. Consequently, the service entity and its auxiliary entities are seen as separate entities.

6.5.3 Interaction signatures

Interaction signatures describe patterns of interaction contributions between two or more behaviour blocks. They also facilitate the visualisation of the events in an interaction relation. Interaction signatures are depicted as a shorthand notation that represent interaction relations.

An interaction signature models the relation(s) between two or more behaviour blocks by making explicit the information passed or exchanged in the interaction, but ignores the conditions and constraints that govern the interaction. That is they describe how two or more behaviours are 'glued together' in a specific way, and what information is interchanged.

To explain the use of signatures we use an example definition. Figure 6.9 shows the interaction relation between two behaviour blocks bb-OptimumRoute and bb-Traveller. Four interactions have been defined I-init, I-rej, I-acpt and I-route. The information attribute (ι) for the i-init and the i-route interactions has been specified as well as the specific constraints.

However what we are interested at this moment is not in how one behaviour block would constraint the other, but what information would be supply as input to realise the service, and what the outcome of the service should be. For this purpose a simplified definition would suffice (see Figure 6.10). With this information one can, later on, model a service that would be adequate for the given case, or use an exiting service model(s) that is compatible with the case under consideration.

Figure 6.10 shows the signature for the preceding example. This definition contains the information exchanged or passed in the identified interactions i-init and i-route, and the interaction types. The interaction i-init, for instance, has been portrayed as an acknowledge negotiation. Interaction types are explained later in this section. Interaction signatures, which are modelled at the behaviour domain, consists of two basic

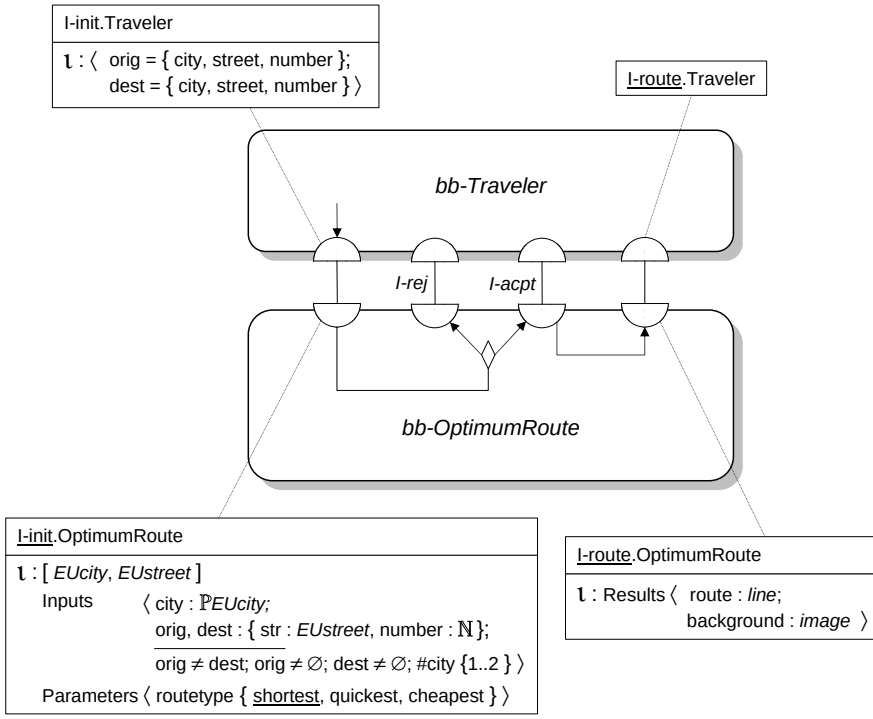


Figure 6.9: Interactions between two behaviour blocks

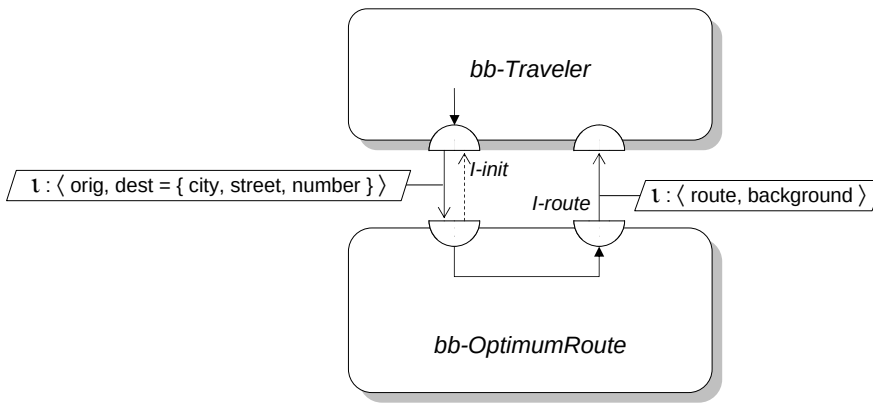


Figure 6.10: Interaction signature

elements: the manipulated information, represented as *items*, and the interaction type.

Figure 6.10 depicts the concept of items. Since the behaviour of a GSI system is accomplished through the creation, use or transformation of some information we introduce the concept of *items* to explicitly show the information on which behaviour is performed. An item may represent a set of parameters, some input, a combination of both or some result.

Items can be coupled to the actions or interactions depicted in a behaviour. The coupling of items distinguishes four different modes: create, use, consume, change or destroy (see Figure 6.11). These modes indicate the type of action that is performed on a coupled item.

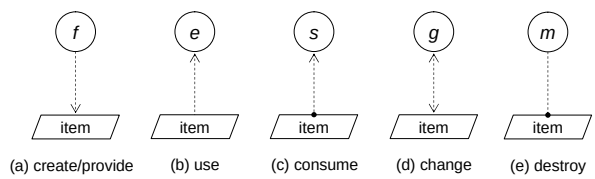


Figure 6.11: Items coupled to actions

Examples of interactions types between two or more behaviour blocks are listed below, and are depicted in the sample models shown in Figures 6.12 and 6.13 respectively. Examples 2 and 3 show both the basic notation and the shorthand notations, the other examples only show the shorthand notations. The basic notation for the remaining cases can be intuitively derived using examples 2 and 3.

Example 1: Unacknowledged service, for example, *BBent-A* passes a message to *BBent-B*, *BBent-B* receives the message and the interaction is completed.

Example 2: Acknowledged service, for example, *BBent-A* sends a message to *BBent-B*, *BBent-B* receives the message and then sends a notification back to *BBent-A*, this notification defines the end of the interaction. This notification does not trigger iterations on the interaction, it just serves to notify *BBent-A* on whether or not the interaction has yield a successful completion.

Example 3: Provision of an item, for example, *BBent-A* executes some operations and generates a feature dataset that is provided to *BBent-B*. *BBent-B* may use the dataset for further processing.

Example 4: Request and response, an associated item is delivered as a result of a specific request, for example, *BBent-A* makes a request for a feature dataset to *BBent-B* including, say, an area of interest, some feature types, and possibly a scale. *BBent-B* processes the request and provides the required feature dataset to *BBent-A*.

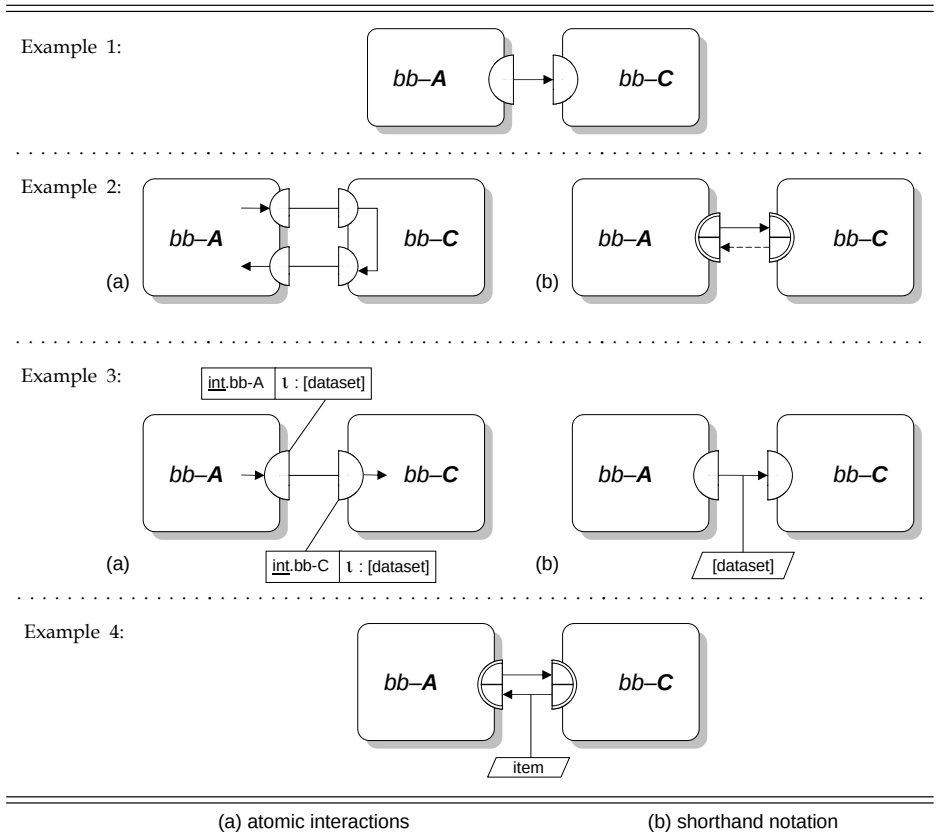


Figure 6.12: Examples of interaction types [1]

Example 5: Cooperative provision of a service, for example, *BBent-A* sends a message to *BBent-C* in coordination with *BBent-B*, which provides an item, say, a dataset that is required by *BBent-C* for processing.

Example 6: Negotiation, for example, *BBent-A* and *BBent-B* send messages to each other back and forth until they agree upon their corespondent constraints, at that moment the interaction is completed, e.g., a login operation. The negotiation may also involved items although that is not shown in the figure.

Example 7: Processing requests of an associated item, for example, *BBent-A* provides *BBent-B* with a feature dataset, *BBent-B* performs some processing on the provided dataset and then gives the resulting dataset back to *BBent-A*.

Although most of the examples show two behaviour blocks, any number of behaviours may participate in a single interaction. Additionally, the behaviours depicted in any of the previous example list may be of a complex nature, and could be refined if

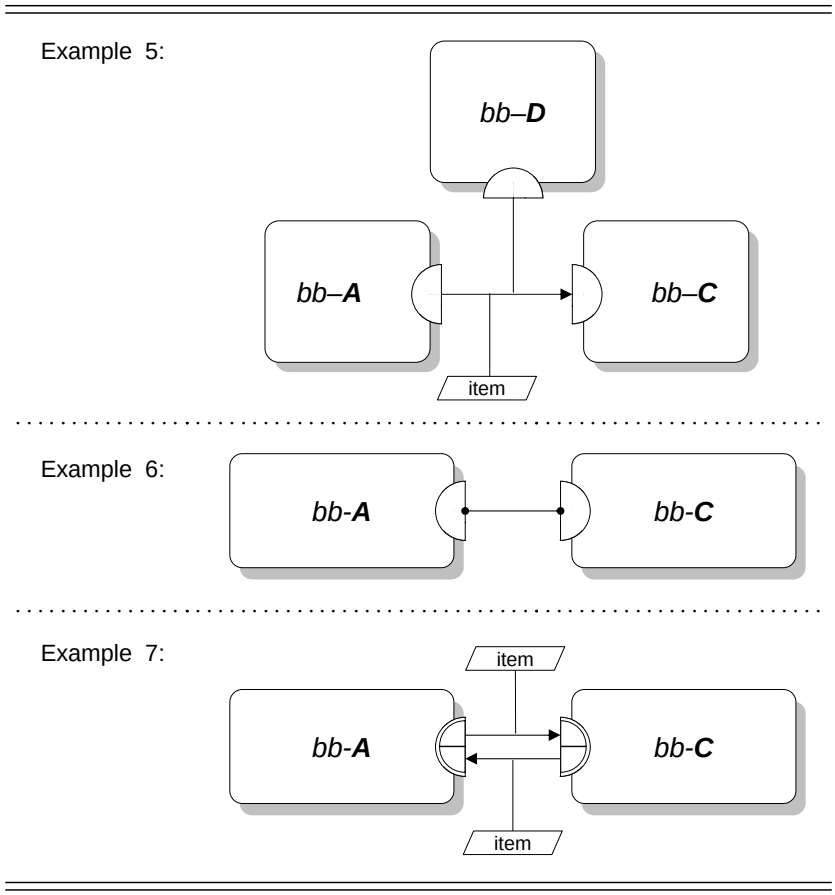


Figure 6.13: Examples of interaction types [2]

required. That means that blocks of behaviour may be decomposed into two or more sub-behaviours with their corresponding interactions.

Due to our design objective of facilitating service chaining, our architectural style defines an important constraint on the structure of interactions. In cases such as those depicted in examples 4 and 5, where a processing entity receives an input, either with or without an associated item, and after some operations delivers an item as a result, the interaction that handle the input and the output should not be modelled as part of a single interaction, but as two separate yet associated interactions.

This approach facilitates the sequential chaining of behaviours, specially in those cases where the behaviour ($\mathsf{B}_{\text{requesting}}$) that triggers the production of a particular item is not the same behaviour that receives the output generated by the processing behaviour($\mathsf{B}_{\text{processing}}$), see Figure 6.14.

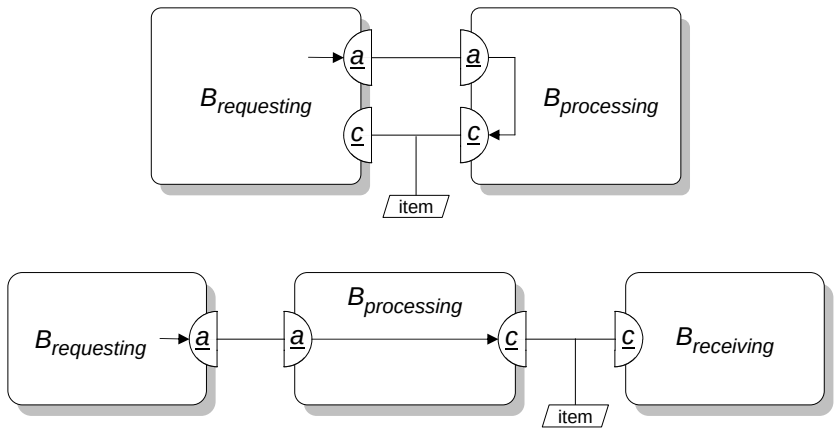


Figure 6.14: Interactions definition for service chaining

6.5.4 Behaviour model

We can now start to identify the set of boundary interactions that fully delimit the TD-service. We call behaviour model, a diagram that captures the interactions between and entity and its environment. This diagram is created by assigning behaviours to entities that participate in a service, and by refining the interconnections between these entities defined in the initial service definition.

At this level, a complete behaviour model includes the boundary level interactions, and their signatures, the ordering relations between these interactions, and if necessary, the items manipulated in the interactions.

For the purpose of refining the initial TD-service model (see Fig 6.7), we present additional detail on the Tax Determination service case. Each taxable piece of land (parcel) is associated with a tax payer. The tax payer is not necessarily the owner of the land, it could be any other person that is currently benefiting from the use of the land, like a tenant. There are multiple factors used in the determination of the tax that is assigned to each parcel, the most important one being the area of the parcel. This area has to be surveyed and the parties involved have to agree on the results of this survey.

Figure 6.15 shows behaviour model of the TD-service definition introduced earlier.

First of all, the diagram of Figure 6.15 shows that the two main entities of the TD-service, *Tax payer* and *Taxation module*, have been replaced by their associated behaviours, *BB-TaxPayer* and *BB-TDservice* respectively. Additionally, Figure 6.15 shows that the interaction point of the TD-service has been replaced by the interactions: *I-reg*, *I-svy* and *I-inv*.

The first interaction, *I-reg*, models the action of a Tax Payer registering into the

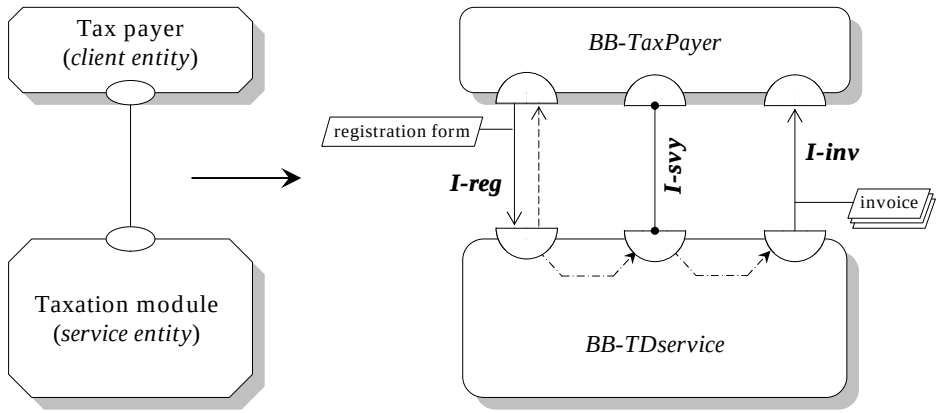


Figure 6.15: refinement of the TD-service definition

tax register. An input item is required for this task, which is modeled by the item *registration form*. The arrows in the interaction describe the type of interaction that is taking place.

The second interaction, *I-svy*, represents the action of surveying the property that the tax payer has registered into the system. This time the interaction type shows that the survey interaction is a negotiation, and only completed when the two participating entities come to an agreement, in this case, on the area and limits of the parcel under consideration.

The last interaction, *I-inv*, models the action of notifying the tax payer on the value that has been taxed to the property or properties associated with him (the tax payer). The item *invoice* models the result of the service. The *invoice* item is shown replicated in Figure 6.15 to show that in turn multiple invoices are sent to the tax payer. The reasons for this are: 1. the tax payer is associated with more than one parcel; or 2. according to the regulations taxes on agricultural land are charged on periodical basis.

In Figure 6.15 the single interaction shown before the refinement, has not been qualified. That is, there are no arrow heads and dots on the extremes of the line connecting the interaction points. This shows that the interaction needs refinement.

The same holds for the relationships between interactions inside the TD-service behaviour block, which are depicted in the figure as dashed lines. However in the later case, although the interaction relations need refinement, arrow heads are provided to show the ordering in which the interactions occur.

For simplicity purposes, not all the details on the interactions of the TD-service have been added to the service definition depicted in Figure 6.15. Interaction constraints, and interaction attributes, for example, are not shown.

Chapter 7

The internal perspective model

Those who do not know the conditions
of mountains and forests, hazardous defiles, marshes, and swamps
cannot conduct the march of an army.

Sun Tzu

In this chapter we concentrate on a development method to produce 'internal perspective models'. These are models that are closer to available implementation mechanisms and refine the more abstract 'external perspective models'. The chapter proposes a design pattern to guide and constrain the definition of architectural elements. The chapter shows how to use design concepts to represent elements of a geo-information system. The chapter also explains the process of specifying services as individual functions or as service chains.

The chapter is structured as follows: section 7.1 explains the principles of service decomposition and the criteria for the definition of architectural elements; section 7.2 introduces the development steps followed to obtain internal perspective models; section 7.3 provides guidelines for the assessing the correctness of proposed development steps. section 7.4 explains techniques to describe services; and section 7.5 shows an example of the creation of internal perspective models.

7.1 Decomposition goals

The objective of the architectural design phase (see section 4.2) is to decompose a system into a set of interacting parts, such that the system's observable behaviour (external perspective model) is implemented by the composed behaviour of these

parts. We use architectural elements, as introduced in section 4.5, to represent system parts.

System decomposition has to be performed according to certain specific criteria, like reusability of elements and the correctness of the resulting composition. In order to make our method concrete, we prescribe the use of a decomposition pattern, referred to as the mediator pattern. Decomposition steps can be performed recursively, until a final decomposition is obtained.

7.1.1 Criteria

The main goal of a decomposition step is to define the internal structure of the required service. This decomposition step should be performed whenever there is no single concrete element that can provide the required service as defined by the external perspective. The Figure 7.1 shows a decomposition step of a required service, which results in a set of related elements. In Figure 7.1 the required service and the resulting set of elements are represented as behaviour blocks. The lines connecting the behaviour blocks of the decomposition result in Figure 7.1 denote the interactions between their corresponding elements.

In the decomposition step depicted in Figure 7.1 we apply available elements, whenever possible, or define new elements in such a way that they can be reused in other development projects. Furthermore, each element should be defined in such a way that it can be later replaced by another element, for example, with the same functionality but with an improved implementation. Figure 7.1 also shows that there are multiple alternative decompositions that implement the required service.

In our method we aim at separating the data sharing concerns from processing concerns, by assigning them to different types of architectural elements, namely data and processing elements respectively. Although this separation is an important criterion for defining architectural elements, we also realise that processing elements may need data that is specific to their functionality and should not be detached from them. In this sense a processing element is self-contained, because it contains all the data resources it needs to properly operate.

For example, one can design an architectural element that can be used to determine a route between two given points through a road network. We can specify the x and y coordinates of the two points as the only input needed. The element uses this input and a road network from an internal storage to find the route between the points. In this case, such an element can only be used to determine routes based on the road network available internally in the element, for instance, the Dutch road network.

Separating data sharing concerns from processing concerns fosters reuse, since it allows processing and data elements to evolve independently. Accordingly, a processing element should be designed to be independent of any context sensitive data, like data that changes over multiple instantiations, or data that can be obtained from the en-

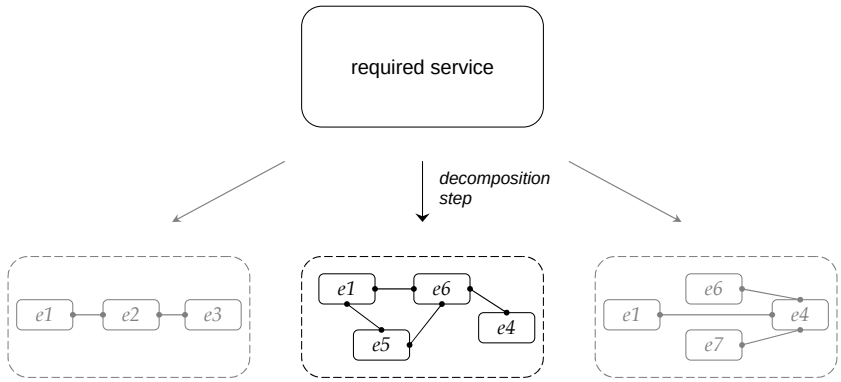


Figure 7.1: Decomposition step

vironment of the element. This separation may also improve maintainability, since it potentially simplifies processing elements.

Applying this separation criterion to the route determination element above, we could design a processing element that determines routes between points, such that this element gets as input the points of interest and a corresponding transportation network. In this case, the element is context-independent and can be used in more situations, such as when the two points are located in different countries. In addition, because the function of the element focuses only on the determination of a route, it can even be used to determine a route along a railroad network, if the railway network is the kind of transportation network given as input.

Consequently, specially in the case of geo-information processing, it suffices to have stateless elements, which are architectural elements that can not make use of any stored data from previous instantiations. This means that all the data needed by the element to operate properly should be modelled explicitly with its interactions, and, therefore, this data should be provided in each instantiation. The separation criteria applies at design level, which means that an implementation may still integrate a processing element and data element, e.g., for efficiency reasons, if these elements belong to the same implementation authority.

The environment of an architectural element consists of other architectural elements and possibly the system's environment. An architectural element should make its service accessible to its environment via interactions. We may have large architectural elements that can provide useful services by themselves, but they still should have the capability of being combined with other (large) architectural elements in the case extended or specialised services are needed. This implies that architectural elements should be defined such that they can be combined with other elements solely on the basis of the specification of their observable behaviour, namely their external perspective model.

7.1.2 Decomposition pattern

Many alternative approaches can be adopted for the realisation of compositions [HBCS03, ACD+03]. Although it is possible to build up a composition wholly from already existing generic services, in our approach we want to centralise the responsibility for the provision of a service in a single element, and, furthermore, we want to shield the use of multiple elements from the user. Therefore, we adopt a structured composition form in which a special element is defined to coordinate the interactions between the other architectural elements, and to provide an interface to the service user. This leads us to the mediator pattern, in which a central element plays the role of a mediator. Figure 7.2 depicts a decomposition that applies the mediator pattern. The mediator pattern may be implemented using mediator replicas in order to avoid that the mediator becomes a single point of failure.

Besides the criteria mentioned above, the adoption of this decomposition pattern serves three objectives:

1. it allows us to define a set of concrete development steps for the design of GSI services, which results in a composition of independently-designed elements;

2. it allows the organisation of a set of services into a behaviour definition that has a single coordinating element, making the service realisation accountable for the user;

3. it facilitates the use of workflow languages to implement the mediator behaviour, which choreographs the use of third-party services.

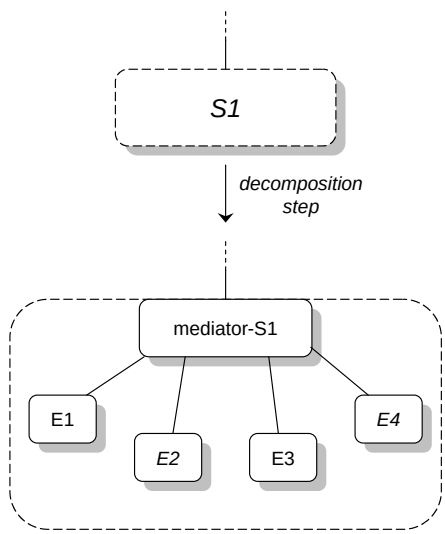


Figure 7.2: Mediated compositions

This pattern is also being used in other architectures for service composition, like, for instance, in compositions of Web Services (see [HBCS03]).

7.1.3 Recursive pattern application

A decomposition step may result in a structure containing architectural elements that do not have a realisation available. In this case we may decide to apply the mediator pattern again to develop these architectural elements. Therefore, we conclude that this decomposition pattern can be applied recursively, until all processing and data elements can be either mapped onto available components or can be created straightforwardly.

Figure 7.3 shows that the mediator decomposition pattern can be applied recursively until proper architectural elements are obtained.

The recursive application of the mediator pattern results in a hierarchy of mediators, each one controlling their corresponding architectural elements and shielding them from a higher level user.

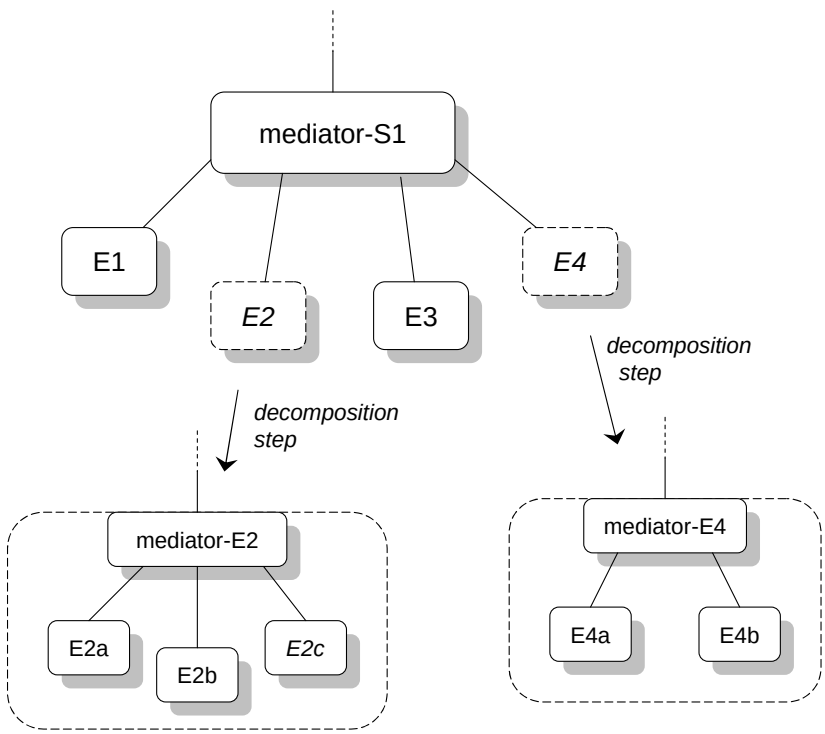


Figure 7.3: Recursive mediated compositions

7.2 Decomposition method

In a decomposition step we start with an extended service definition according to the external perspective, and end up with a set of interconnected elements that define the internal perspective of the required service.

7.2.1 Overview

We define a method for the development of the internal perspective of a service that consists of the following steps (see Figure 7.4):

1. Transformation to integrated form: in this step the required service definition is transformed to the integrated form (see section 5.3.2), by abstracting from the distribution of responsibility for interactions between the service provider and the service users;

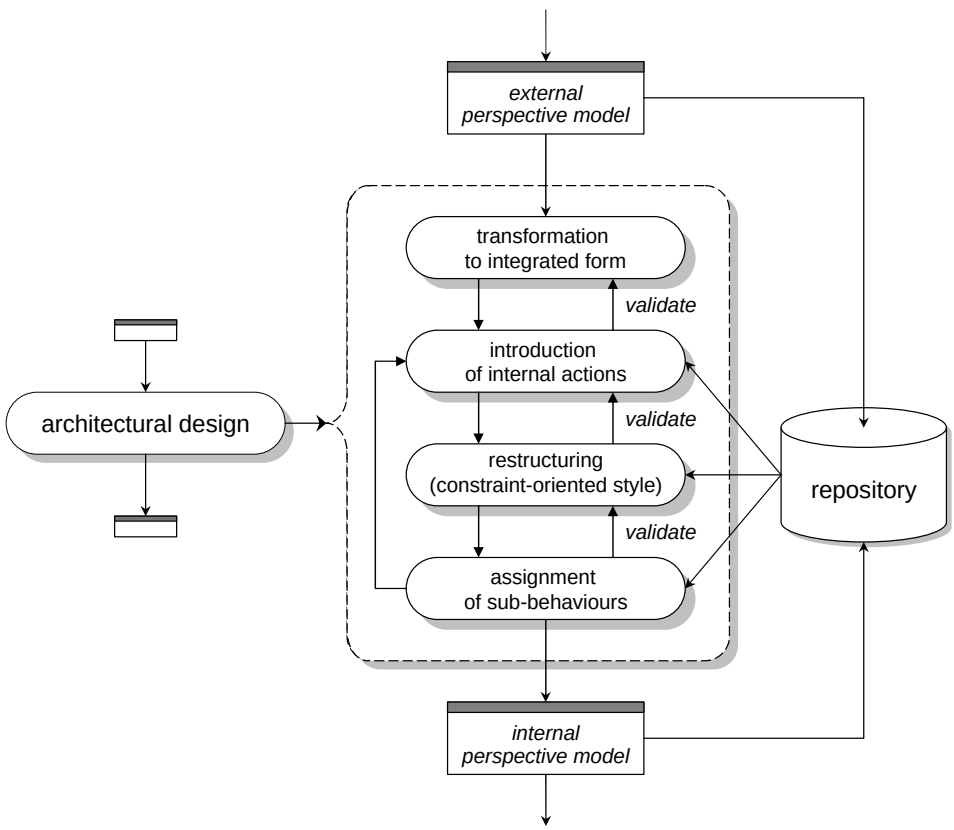


Figure 7.4: Decomposition method

2. Introduction of internal actions: in this step the behaviour of the service in integrated form is refined by introducing internal actions;
3. Restructuring: in this step the refined behaviour is restructured according to the constraint-oriented style (see section 5.4, i.e., such that different constraints on actions can be separated in sub-behaviours;
4. Assignment of sub-behaviours: in this step sub-behaviours of the restructured specification are assigned to elements.

Since we want to obtain a decomposition that consists mainly of reusable elements from a repository, step 2 should be performed hand in hand with the identification of elements that can be part of the required service. This means that we choose internal actions that on one hand relate to parts of the total functionality of the required service, but on the other hand can be supported by available elements.

The final composition of elements that characterise the internal perspective has to conform with the external perspective, which implies that each step in our decomposition method has to be validated for correctness.

Figure 7.5 illustrates the first step of our decomposition method. We consider in Figure 7.5 a fictitious required service that can be described as a behaviour consisting of interaction S followed by interaction F. In this step the interaction contributions for S and F are combined, so that we obtain a behaviour consisting of actions S and F, respectively. This step is necessary because we may decide at a later stage to distribute the interaction responsibilities in a different way than originally done in the required service.

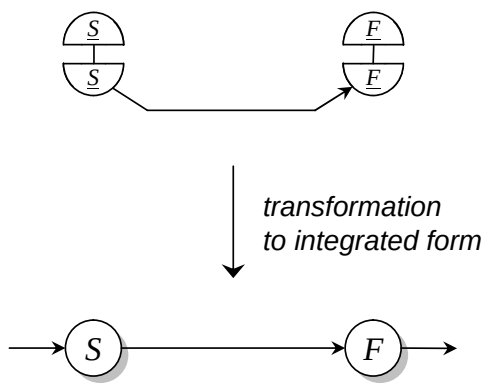


Figure 7.5: Step 1: transformation to integrated form

Figure 7.6 illustrates the second step of our decomposition method. In the example in Figure 7.6 actions a,b,c and d are inserted in the integrated behaviour, resulting in a more detailed behaviour definition. The choice for these specific actions has been dictated by the availability of elements that can perform these actions. For the sake

of conciseness we assume that our architectural elements perform only one action, but in practice they have much more complex behaviours.

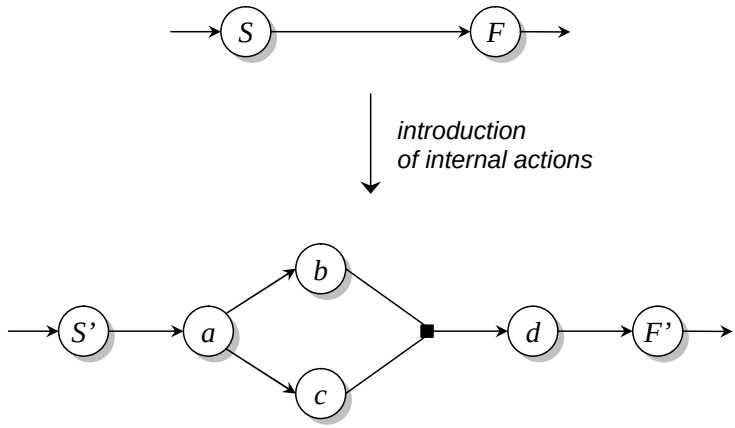


Figure 7.6: Step 2: introduction of internal actions

Figure 7.7 illustrates the third step of our decomposition method. In Figure 7.7 the responsibilities for performing actions have been split, resulting in a behaviour that is structured according to the constraint-oriented style.

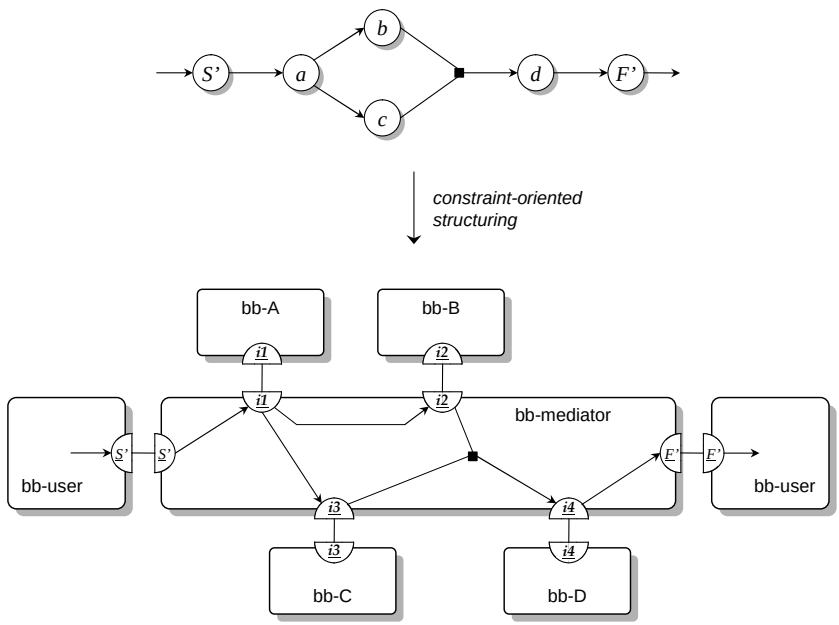


Figure 7.7: Step 3: constraint-oriented restructuring

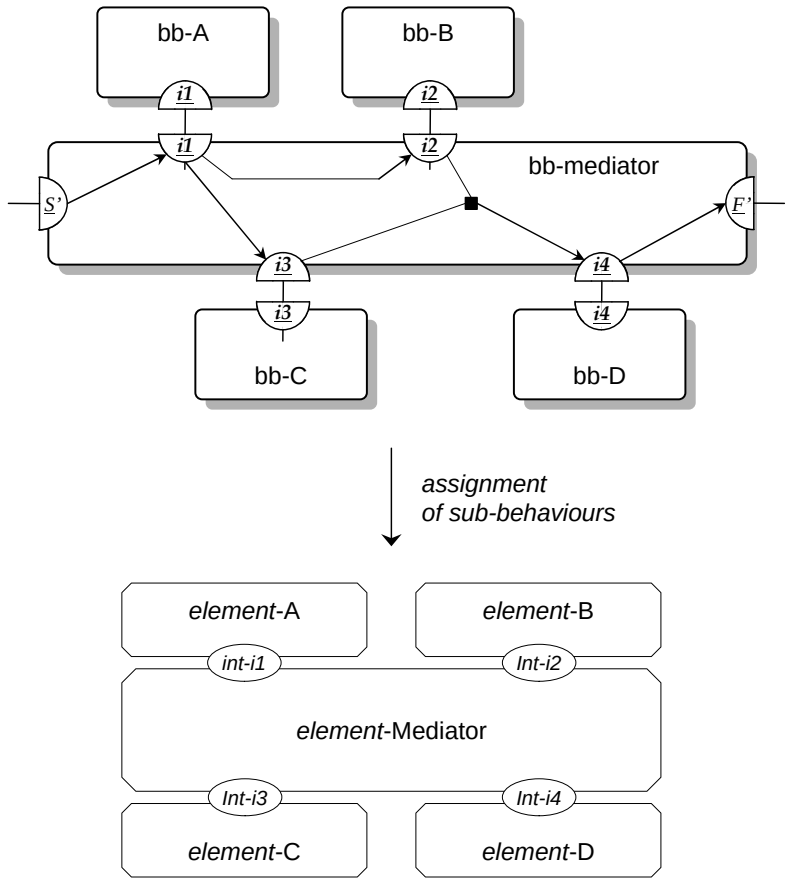


Figure 7.8: Step 4: assignment of behaviours to elements

In the fourth step of our decomposition method, each sub-behaviour is assigned to an element or to the service user. In Figure 7.8 we show that behaviours bb-A, bb-B, bb-C and bb-D are assigned to elements Element-A, Element-B, Element-C and Element-D respectively, and that behaviour bb-mediator is assigned to the mediator Element-Mediator. In Figure 7.8 we omit the service users.

7.2.2 Introduction of internal behaviour

The most important step in our decomposition method is the introduction of internal actions. This step is difficult to automate and is subject to designers choice, interpretation and creativity. A behaviour may be decomposed in a number of ways, by adding different alternative internal actions. Specific technical reasons normally determine the decomposition that is chosen in a refinement step.

To identify the actions that can be inserted, one analyses the required service and defines a functional partition of this service. This partition determines the functions necessary to realise the required service. They should be identified taken into account the functionality offered by the available elements from the repository. Once these functions are isolated and represented as actions in the refined behaviour, indirectly a set of elements is selected for usage. Some of these elements may have already have an implementation that can be used in the realisation of the required service, but some others may have to be built.

The choice of internal actions should also be done with the behaviour of the future mediator in mind, or a mediator behaviour may appear naturally when sub-behaviours are identified as shown in the example of Figure 7.7. If this is not the case, the introduction of internal actions step should be revisited such that direct relations between sub-behaviours that will correspond to data and processing elements are replaced by indirect relations via the mediator behaviour.

When assigning sub-behaviours to elements (step 4 of our method), the sub-behaviours have to match the behaviours of the elements in the repository (external perspective). This is normally not the case, which implies that the introduction of internal actions may have to be reconsidered, so that the desired matching of behaviours can be obtained.

7.2.3 Composition structures

The way in which the required service is internally structured depends on the decision as to how to move the data or results that are exchange between elements. Since we use the mediator pattern to structure service definitions, there are two approaches to define the interactions of a mediator with other architectural elements: 1) Make each element interact with the mediator, both for the purposes of control and data exchange; or 2) interact with the mediator for control and directly for data exchange.

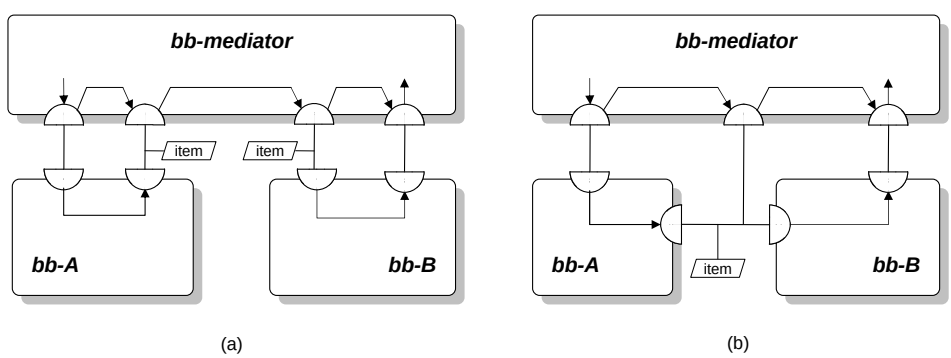


Figure 7.9: Composition structures

Figure 7.9 depicts these two different composition structures. Figure 7.9a shows the structure for the first case, where all control and data (represented as an item in the figure) flows are realised through the mediator. Figure 7.9b shows the structure for the second case, where only the control is coordinated by the mediator but the data is allowed two be exchanged directly between processing elements.

7.3 Correctness assessment

Correctness assessment is used to validate the refinement of an abstract behaviour (e.g., an external perspective model) into a number of interrelated fine-grained behaviours (e.g., an internal perspective model) that conform to the behaviour provided by the abstract behaviour. We refer to a behaviour and its corresponding refined specification as abstract and concrete behaviours respectively. A concrete behaviour is correct with respect to an abstract behaviour if it preserves all properties of the abstract behaviour.

In a correctness assessment process, the activity units (actions and interactions) that are modelled in the abstract behaviour are called *abstract activity units* (abstract action and abstract interaction), while the activity units modelled with the corresponding concrete behaviour(s) are called *concrete activity units* (concrete action and concrete interaction). Abstract behaviours are exhibited by system parts, which are called abstract system parts.

Two main requirements have to be observed when refining an abstract behaviour into a concrete behaviour [Qua98, Sin95, FP94]:

- *Preservation of relations*. The relations between actions in the abstract behaviour should be preserved by the relations between their corresponding actions in the concrete behaviour.
- *Preservation of action values*. The values established in actions in the abstract behaviour should be preserved by the values established in their corresponding actions in the concrete behaviour.

Figure 7.10 shows the correctness assessment method that we adopt. The left of the figure depicts an abstract behaviour, represented as an oval, that is refined into a concrete behaviour, represented as circles. Relations between concrete behaviours are represented as straight lines. The right of the figure shows an abstracted behaviour that is obtained from a composition of the concrete behaviour.

To assess the correctness of the refinement of an abstract behaviour into a concrete behaviour, our method has the following steps [Qua98, FP94]:

1. Identification of reference activity units and inserted activity units in the concrete behaviour. *Reference activity units* are activity units in the concrete be-

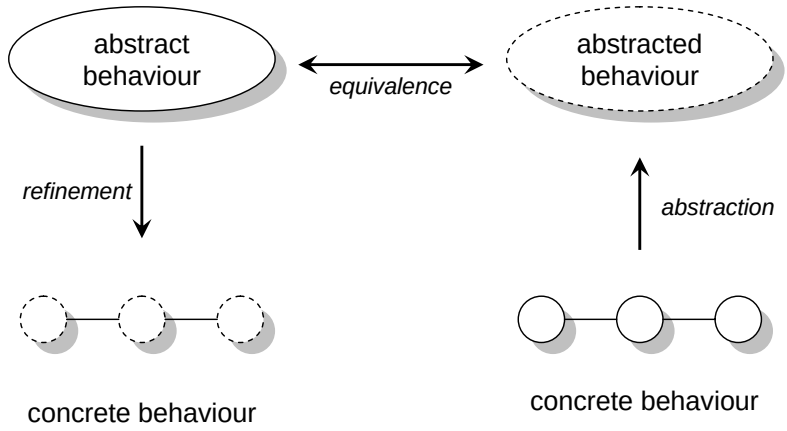


Figure 7.10: Correctness assessment

haviour that correspond to activity units in the abstract behaviour. *Inserted activity units* are activity units in the concrete behaviour that do not correspond to activity units in the abstract behaviour, i.e. they are activity units that have been inserted during refinement;

2. Abstraction from all of the inserted activity units. This requires for each inserted activity unit (action) i: a) the replacement of each causality condition in which i occurs by an equivalent causality condition in which i does not occur; b) the replacement of each reference relation defined in terms of values established in i by an equivalent reference relation in which these values do not occur.

3. Replacement of each group of reference activity units by an abstract activity unit. Each group of reference activity units whose occurrence corresponds to the occurrence of a single abstract activity units should be replaced by the abstract activity unit.

4. Comparison of the abstraction of the concrete behaviour to the original abstract behaviour. The concrete behaviour only conforms to the abstract behaviour if the abstracted concrete behaviour and the abstract behaviour comply to a certain correctness relation. This correctness relation can be:

   - an equivalence relation which defines that the abstracted concrete behaviour should preserve all properties of the abstract behaviour (e.g., for action values this means that all action values possible for an abstract action are also possible for the corresponding concrete actions), or
   - a partial order relation which defines that the abstracted concrete behaviour should preserve a subset of the properties of the abstract behaviour (e.g., for action values this means that a subset of the action values possible for an abstract action are also possible for the corresponding concrete actions).

The proposed strategy, especially step 3, assumes that in the behaviour refinement process each abstract activity unit is replaced by a single concrete reference activity unit. This means that when we perform the introduction of internal actions in the decomposition method (see section 7.2.2), all issues related to interface refinement have been considered before.

This strategy is based on the recognition that the refinement of an abstract behaviour can result in many alternative concrete behaviours, whereas the abstraction of a concrete behaviour is unique. Therefore, the method requires the assessment of conformance by comparing the abstraction of the concrete behaviour to the original abstract behaviour.

Consider for example, the diagram depicted in Figure 7.11. The model depicts the abstract behaviour *bb-A*, which includes four interactions represented by: int1, int2, int3 and int4. The model states that int1 is followed by int2, and that int2 is followed by int3 and int4.

We have various possibilities for the refinement of the abstract behaviour *bb-A* into two more concrete behaviours, *bb-A1* and *bb-A2*. We have chosen two alternative scenarios for the assignment of the interactions from the abstract behaviour to the more concrete behaviours. In the first scenario the reference interaction corresponding to int1 is assigned to *bb-A1* , while the reference interactions corresponding int2, int3 and int4 are assigned to *bb-A2*. In the second scenario, the reference interactions corresponding to int1 and int4 are assigned to *bb-A1*, while the reference interactions corresponding to int2 and int3 are assigned to *bb-A2*.

Figure 7.12 shows the representation of the refinement of the abstract behaviour *bb-A* according to the first assignment scenario. Interactions int1', int2', int3' and int4' are reference interactions that correspond to int1, int2, int3 and int4. Interaction int5 is an inserted interaction in the concrete behaviour.

Interaction int1' has been assigned to the concrete behaviour *bb-A1*, while interactions int2', int3' and int4' have been assigned to the concrete behaviour *bb-A2*. According to the abstract definition, interaction int2 is only allowed to occur after the occurrence of int1. This relation has to be kept after the assignment of the concrete reference

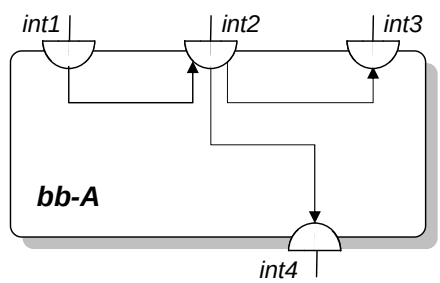


Figure 7.11: Abstract behaviour

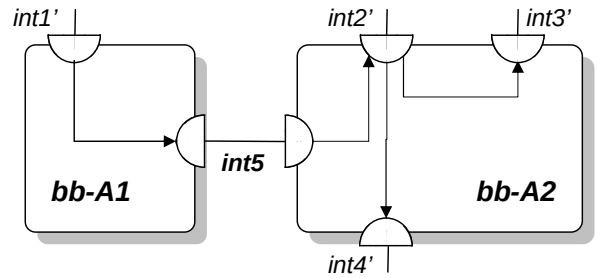


Figure 7.12: Refinement of an abstract behaviour into two concrete behaviours

interactions to the concrete behaviours. Therefore, there should be a relationship between interactions int1' and int2', which are now assigned to separate behaviour blocks. This relationship is accomplished by introducing interaction int5, which defines a constrained relation between the two concrete behaviours *bb-a1* and *bb-A2*. As a result int1' is followed by int5, which is followed by int2'.

The refinement of the abstract behaviour *bb-A* according to the second assignment scenario is shown in Figure 7.13. In this case, interactions int1' and int4' have been assigned to concrete behaviour *bb-A1*, and interactions int2' and int3' have been assigned to concrete behaviour *bb-A2*.

In this refinement, interactions int1' and int2' as well as int2' and int4' have to be related. This relations are kept by introducing two interactions int5 and int6. These interactions allow behaviours *bb-A1* and *bb-A2* to exchange information that was previously centralised.

For simplicity reasons we have not introduced internal activity units, neither have we defined sub-behaviours for the concrete components. According to the correctness assessment steps introduced earlier, to assess whether or not the abstract behaviour was correctly refined, we have to merge the concrete behaviours, abstracting from

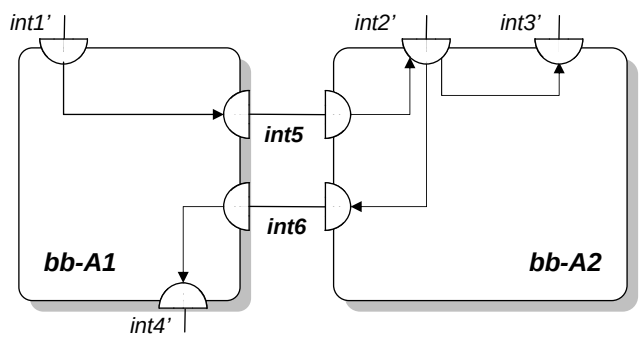


Figure 7.13: Alternative refinement of an abstract behaviour into two concrete behaviours

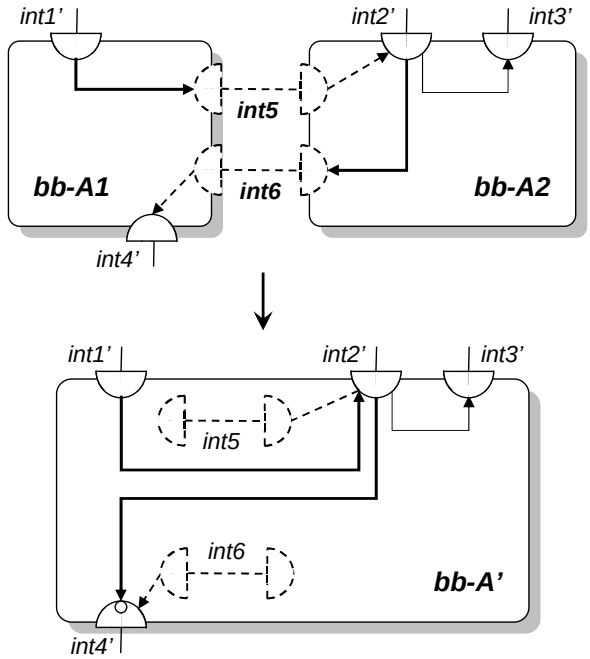


Figure 7.14: Alternative refinement of an abstract behaviour into two concrete behaviours

inserted interactions, to produce an abstracted behaviour than can be compared with the original abstract behaviour.

In the case of the second scenario, when we abstract from the inserted interactions that relate the two concrete behaviours *bb-A1* and *bb-A2*, we have to remove interaction int5, and then make the condition for interaction int5, which is int1', a condition for interaction int2'. Similarly, we remove interaction int6, and make its condition, which is int2', a condition for interaction int4'. The result is then an abstracted behaviour that can be compared to the abstract behaviour (see Figure 7.14).

7.4 Service descriptions

A GSI service is an accessible piece of functionality that requires an adequate description to facilitate its discovery. These service descriptions should convey enough information such that one can find and use required services. Service descriptions have two main sections, a general section that applies to all services, and a specific section that depends on the nature of the elements that provide the service.

The general metadata section is explained in detail in appendix A.3. The following two sections describe the specific issues on service description for data and processing

elements, respectively.

7.4.1 Data descriptions

Data elements represent sources of data. Examples of data elements include a document, an image, a data collection, application specific information (e.g., today's traffic data). In GSDM awareness about the nature and characteristics of the data represented is of primary importance, because data need not only to be discovered but also combined and transformed. Data may also need to be moved from the location where it is stored to the location where it is used, normally by, a processing element.

Metadata elements are used to describe geographic data. Metadata elements, also called referential attributes, provide a meaningful description of the data, which facilitates its discovery, access and use. Metadata enables one to determine whether particular geographic data is useful for a particular application. We use metadata to describe collections (complete datasets), individual geographic objects (geoObjects), themes, and composites. This metadata describes the semantics of the data and prescribes the form in which this data can be exchange. Figure 6.5 in section 6.4 shows the relation between geographic features and metadata elements. A geographic feature can contain a number of metadata elements that convey its descriptive information.

Figure 7.15 shows the metadata schema for describing geographic data, which defines a set of metadata elements. Typically only a subset of the elements showed in Figure 7.15 is used. This subset of the elements is known as *core metadata* since it comprises the minimum metadata elements required to identify a dataset. These elements are the Identifier and the Creator. Other elements, like the TemporalResolution or CollectionDate are not considered core elements, and as such they are not compulsory but they should be provided, whenever possible, in order to increase the potential of

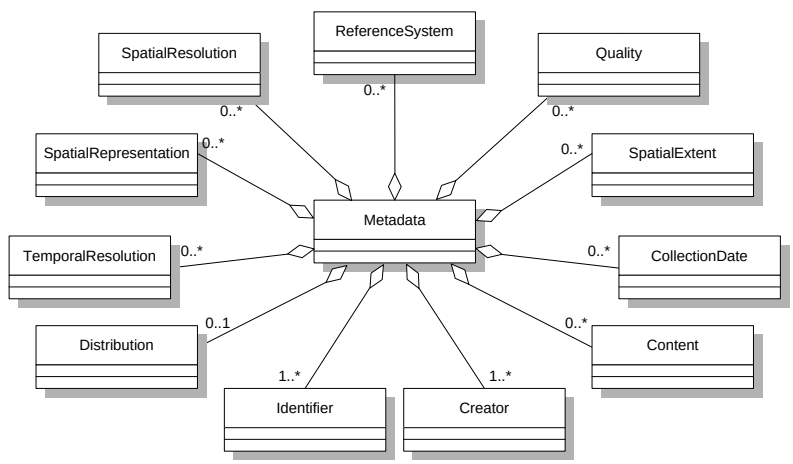


Figure 7.15: Metadata elements

re-usability of the data.

7.4.2 Processing descriptions

Processing services perform actions on data, that is, they use, modify, create or destroy data resources as they carry out their intended tasks. Processing services represent distinct, identifiable and independent units of processing.

These units of processing could address a multiplicity of operations of different nature. In order to properly structure these service descriptions, we provide a semantic classification of services into service groups. We based the classification on the OGC service taxonomy domains [OGC02, OGC03b]. The classification presented here is used to facilitate the organisation, discovery and manipulation of processing services .

We organise our service groups based on the structure of the content that they operate on (e.g., features, images), the type of contents either consume or produce (e.g., roads, parcels), the spatial and temporal resolution of these contents (e.g., 6 months, two years, or 1:2000, 1:500), and other descriptors such as area of validity, contents sources, quality of service, etc. We identify five different service groups:

1. *Retrieval services*, which provide access to collections of data or geodata in repositories and databases, i.e., feature access services, coverage access services, sensor collection services and archive services;
2. *Value-added services*, which operate on (geo)data and add value to it. They can perform large computations and transform, combine, or create new (geo)data, i.e., coordinate transformation services, geocoder services, gazetteer services, geoparser services, reverse geocoder services and route determination services;
3. *Visualisation services*, which enable the rendering and portrayal of information, e.g., given one or more inputs, they produce cartographically portrayed maps, perspective views of terrain, annotated images, views of dynamically changing features in space and time, etc.;
4. *Application services*, which support managing user interfaces, graphics, multimedia, and presenting compound documents, i.e., data association and cross-referencing, imagery exploitation, location-based services;
5. *Catalogue services*, which are used to classify, register, describe, search, maintain and access descriptions of services available on the infrastructure.

These descriptions are illustrate the specific functionality provided by a service, and consequently can be used for searching, finding and possibly selecting suitable services. The actual service definition of each of these services in terms of the external and internal perspectives of the service is stored and structured according to the repository (see section 4.4). The schema for this repository is shown in Appendix B.

7.5 Design example

Figure 7.16 shows the EP model of the *Simple Route* service. The model includes two behaviour blocks identified as bb-Traveller and bb-SimpleRoute. The two behaviour blocks have a constrained relationship defined by the interactions I-init and I-route. The model also describes the items exchanged by the two behaviours. This model is used in this section to illustrate the development of internal perspective models.

The internal perspective of a required service is developed through the successive refinement steps that aim at the replacement of the abstract behaviour of the required service by a concrete behaviour specification (see section 7.2).

Figure 7.16 shows a causality relation between the interaction contributions I-init and I-route of the behaviour bb-SimpleRoute.

We start by transforming the specified interactions to their integrated form, which means abstracting from the individual interaction contributions of the involved en-

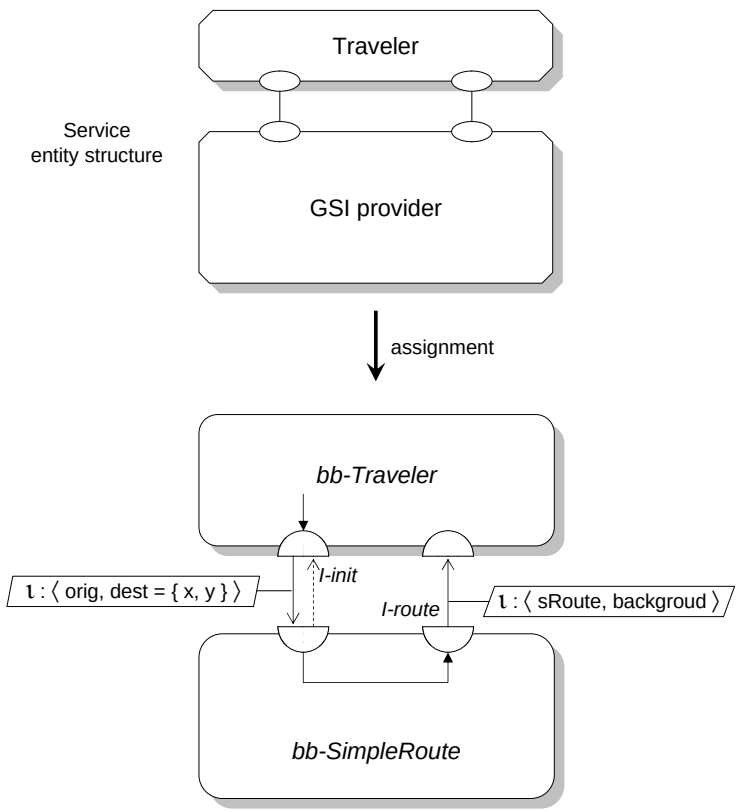


Figure 7.16: Simple route service EP model

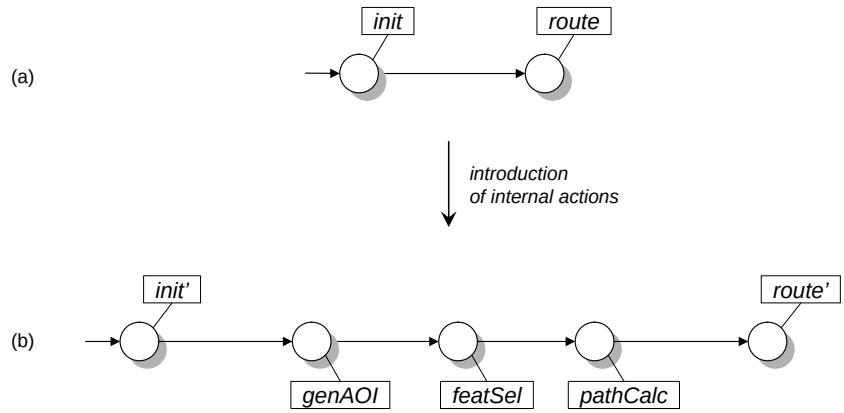


Figure 7.17: Introduction of internal actions to the SimpleRoute behaviour

tities. Figure 7.17a shows the abstract behaviour that models the function of the service provider in an integrated form. It consists of two actions init and route, with init being a condition for route. The textual notation of this behaviour is as follows:

$$\begin{aligned} \mathit{SimpleRoute} : \{\, & \surd \rightarrow \mathit{init}\langle \mathsf{orig},\ \mathsf{dest}\rangle, \\ & \mathit{init}\langle ... \rangle \rightarrow \mathit{route}\langle \mathsf{path},\ \mathsf{backg} \mid \mathsf{sRoute} \mapsto \mathit{fbound}(\mathsf{orig},\mathsf{dest})\rangle \,\} \end{aligned}$$

7.5.1 Introduction of internal actions

In order to identify the different data and processing elements necessary to realise the required service, as defined by the external perspective model, we apply stepwise refinement to the abstract behaviour to determine internal service actions and their respective causality relations.

For the purpose of the refinement process we shortly explain the service walkthrough. Three major steps can be identified for the determination of the route between two points. First identification the geographical area where the two points (origin and destination) are located, second obtention of the road network that correspond to that area, and third, determination of a route between the points.

We this knowledge we can refine the abstract behaviour by introducing three actions, viz. the internal actions genAOI, featSel, and pathCalc. Figure 7.17b depicts the resulting behaviour. Action genAOI models the identification of the area of interest, that is the geographical zone that contains the origin and destination points. Action featSel models the extraction of the features of the transportation network to be used in the calculation. Action pathCalc models the actual determination of the route between the given points.

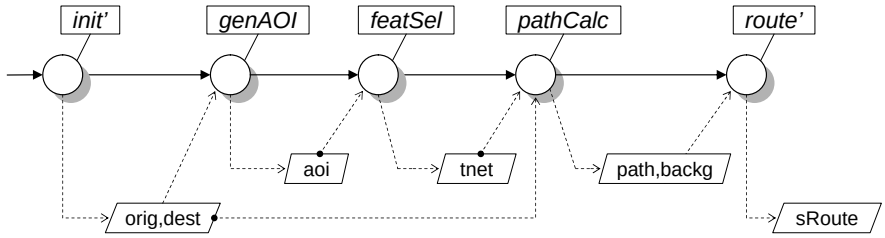


Figure 7.18: Item representation of the SimpleRoute behaviour

Actions genAOI, featSel and pathCalc represent a distribution of part of the processing required to perform the route calculation function. This processing is controlled by the attribute constraints of action route'. These constraints are the responsibility of the service provider and therefore should be enforced in the refined behaviour. Consequently, the refined behaviour in this case could be defined as follows:

$$
\begin{aligned}
\mathit{SimpleRoute} : \{\ & \surd \rightarrow \mathit{init'}\langle \mathsf{orig},\ \mathsf{dest}\rangle, \\
& \mathit{init'}\langle ... \rangle \rightarrow \mathit{genAOI}\langle \mathsf{aoi},[\mathsf{orig},\mathsf{dest}] \mid \mathsf{aoi} \mapsto \mathit{contains}(\mathsf{orig},\mathsf{dest})\rangle, \\
& \mathit{genAOI}\langle ... \rangle \rightarrow \mathit{featSel}\langle \mathsf{tnet},[\mathsf{orig},\mathsf{dest},\mathsf{aoi}]\rangle, \\
& \mathit{featSel}\langle ... \rangle \rightarrow \mathit{pathCalc}\langle \mathsf{path},\mathsf{backg} \mid \mathsf{path} \mapsto \mathit{fbeg}(\mathsf{orig}),\mathsf{path} \mapsto \mathit{fend}(\mathsf{dest})\rangle, \\
& \mathit{pathCalc}\langle ... \rangle \rightarrow \mathit{route'}\langle \mathsf{sRoute},[\mathsf{backg}] \mid \mathsf{sRoute} \mapsto \mathit{fnodes}(\mathsf{path})\rangle \qquad \}
\end{aligned}
$$

The causality relation of the abstract behaviour is preserved in the refined behaviour since the latter defines that init' is a condition for genAOI, genAOI is a condition for featSel, featSel is a condition for pathCalc and pathCalc is a condition for route'. The concatenation of functions should yield the same result. If $\mathit{fbound} = \mathit{fnodes}.(\mathit{fbeg},\mathit{fend})$, then the attribute values of the abstract behaviour are preserved in the refined behaviour. This can be validated by substituting references to values of network and calc by their values or constraints. By applying this rule to the values of route we conclude that sRoute = fnodes(path) can be substituted by sRoute = $\mathit{fnodes}(\mathit{fbeg}(\mathsf{orig}),\mathit{fend}(\mathsf{dest}))$.

The textual notation represents actions in italics followed by a list of attributes and constraints between diagonal brackets. Variables to contain attribute values established by the action are simply separated by commas. Variables of previous actions that are retained by an action (and thus can be referred to by subsequent actions) are shown between square brackets. Attribute constraints are separated by a vertical bar. Functions are shown in plain italics, followed by a list of parameters between round brackets.

Figure 7.18 shows the graphical representation of the attribute information manipulated in the SimpleRoute service represented in terms of items.

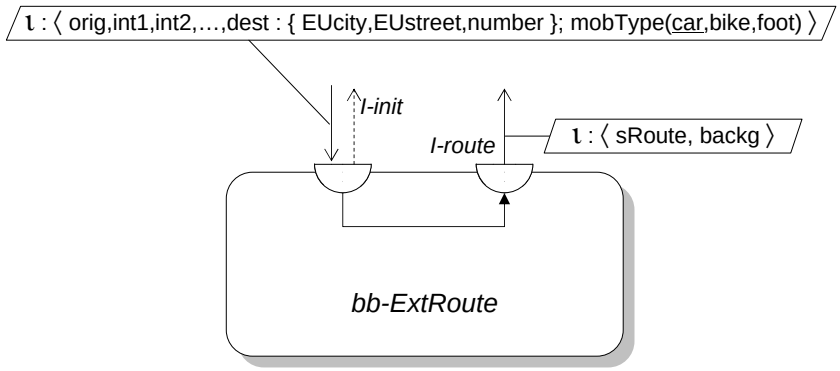


Figure 7.19: Extended route service EP model

7.5.2 Restructuring

Up to now, we have analysed the simple route problem as a single service that accepts two points and determines a route between these points. However, to increase the degree of reusability of the service, we can study similar types of services and introduce into the design additional elements that could handle any extra needed functionality.

To illustrate this we modify the initial version of the *route determination* service. Figure 7.16 showed the external perspective of the SimpleRoute service. The model includes the interaction I-init associated with an item that represents the information that is needed to initiate the service. In that case the information attribute refers to two points as the only data required for the service. These points are the *origin* and the *destination*, modeled as orig and dest, which are defined as pairs of x and y coordinates.

Although, pairs of x and y coordinates is what is required to determine a route, the geographical awareness of most users (travelers) is normally expressed in terms of addresses and not coordinates. This represents a limitation to the usability of this service. Additionally, there could be cases in which intermediate points along the route might also be of interest for the user.

To address this scenarios we introduce a service with a higher degree of reusability called the *extended route* service. Figure 7.19 shows the external perspective model of the bb-ExtRoute service. This model also defines the interaction I-init, which refers to the origin and destination points, modeled as orig and dest as well, but in this case the origin and destination points are defined in terms of addresses, that is a *city name*, a *street name* and a *number*, and nor x and y coordinates. In addition, interaction I-init refers to a serie of intermediate points that should be included in the route determination. Moreover, interaction I-init refers to yet another value, the so-called *mobility type* modeled as mobType, which represents a parameter that specifies the type of transport means to be used in order to go from the origin to the destination.

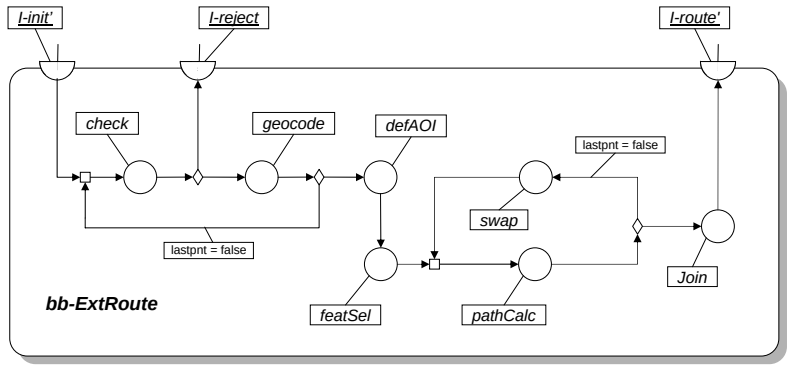


Figure 7.20: Extended route service process flow

This new ExtRoute design could be based on the same set of actions defined with the SimpleRoute service, given that the *addresses* of the later case are converted into the x and y coordinates suitable for the former case. Figure 7.20 shows the behaviour definition of the ExtRoute service as a single function.

This behaviour definition contains a series of actions and causality relations that complement the SimpleRoute service definition presented earlier. Action check for example, models the verification of whether an AddressPoint value is contained within the cities that limit the geographical scope of the service. Acceptable addresses should belong to the subset of European cities as defined by the class EUcity. Action geocode models the transformation of an AddressPoint value into a RoutePoint value. That is, it converts a valid *address* into a pair of x and y coordinates.

The behaviour definition also includes a series of loops, which in this case are introduced as an extension mechanism of the service. For instance, interaction I-init is followed by action check, which in turn is followed by action geocode. After action geocode a *choice* '◊' appears with an *attribute causality condition* defined as lastpnt = false. This condition makes the process iterate until all given AddresPoints have been checked and converted to coordinates. This basically allows for the checking and geocoding of a list (more that two) of points, which can be used to determine a route with multiple vertices. This enables the service to accept not only initial and final points, but also intermediate points.

In addition to the general process flow we also define the different data elements that form part of the service. Here we can make use of both *primitive data types*, such as, integer, char, etc., and *defined data types*, such as, geoObject, theme, etc. (see Section 6.4). Figure 7.21 shows a simplified class diagram that represents some of the data-elements of the ExtRoute service. The diagram includes one aggregated class Route that represents the result of the service. Each instance of this data element encloses the set of streets that form a route.

The diagram also defines that a given *address point*, which complies with the defined

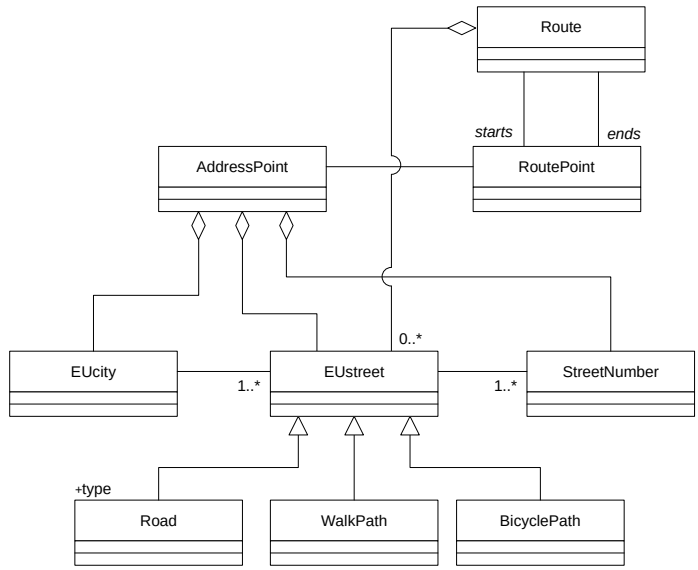


Figure 7.21: Data elements diagram of the extended route determination service

structure of the class AddressPoint, is associated with a *route point* that is part of the class (RoutePoint). Furthermore, a *route point* is part of a *route*, and it can be either the origin point, the destination point or an intermediate point of the route.

These associations, however, apart from representing the relations between the data classes, they represent processing associations between classes as well. That is the associations are realised through the transformation of data. For example a Routepoint result from applying a *geocoding* operation to an Addrespoint.

7.5.3 Assignment of sub-behaviours

From the behaviour definition of the service ExtRoute shown in Figure 7.20 we can discriminate phases as explained in section 7.2.2. This enables the introduction of the mediator element, and the assignment of independent services to separate architectural elements if applicable. In this case, for example, five distinctive independent phases are recognised as necessary to realise the service: 1) Validation of input points. 2) Transformation of input points. 3) Identification and retrieval of the features, in this case road features. 4) Determination of the actual route. 5) Graphic representation of the results.

Figure 7.22 shows the specification of the ExtRoute service with the inclusion of a mediator. In this case we have called the mediator element *Routing*. In addition out of the five recognise phases, only two other independent elements have been identify, namely, *Geocoding* and *Feature selection*. The behaviours of these elements

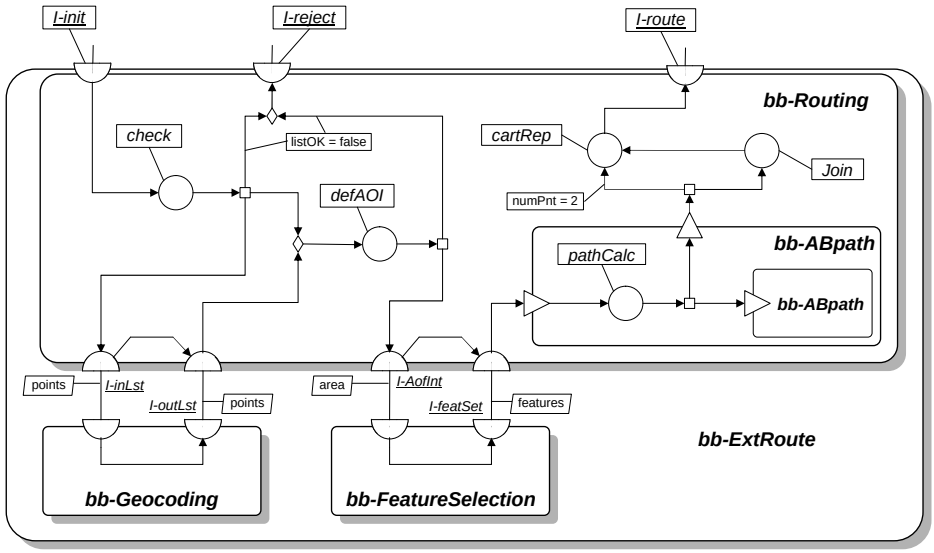


Figure 7.22: Restructured specification of the extended route service with a mediator

are represented by the behaviour blocks bb-Routing, bb-Geocoding and bb-FeatureSelection respectively.

The model depicted in Figure 7.22 presents only a limited number of design elements, such as action, constraints, etc., in order to keep the diagram simple and understandable.

Figure 7.22 illustrates the interaction contributions of the mediator with the other elements. Behaviour block bb-Routing and behaviour block bb-Geocoding interact on I-inLst and I-outLst. Their interaction takes place in case the input points are given in form of *address points*. This means we have defined the service such that it can handle requests with input points given as addresses or as pairs of x and y coordinates.

Behaviours bb-Routing and bb-FeatureSelection interact on I-AofInt and I-featSet. These two behaviours interact in order to retrieve the road network to be used for the route determination from an external data collection. The definition of which features to retrieve, that is *roads*, *walk paths* or *bicycle paths*, is defined by the parameter mobtype.

As part of the realisation development step, the remaining phases (validation, route determination and representation), were assigned to the mediator. Additionally, within behaviour block bb-Routing a causality relation is added between interaction contributions I-inLst and I-outLst. This relation allows interaction I-outLst to refer to the information attributes values established by interaction I-inLst. This relation allows the mediator, bb-Routing, to retain the local constraints and to enforce them on the resulting service provided by bb-Geocoding. The same holds for the interactions between bb-Routing and bb-featureSelection.

A fourth behavioural element, called bb-ABpath, has been defined in order to handle the calculation of the various sections of a route in the case where intermediate points are provided. This last behaviour block however, is not designed as an independent element, but as an internal sub-behaviour that belongs to behaviour bb-Routing because it only represents a repetitive task that in this case only concerns the mediator.

The service example portrayed in Figure 7.22 describe the usefulness of separating elements of a service that are independent from each other. For instance, behaviour block bb-featureSelection represents any third-party service that retrieves a set of vector features, road features in this case, from a data collection, and makes it available for the bb-routing behaviour. This should enable the service to be specialised for different geographical areas, other than Europe, by just replacing the *Feature Selection* service with another service that provides road features of South America, for instance. The textual specification of the ExtRoute service is as follows:

service element △bb-ExtRoute
attributes
...
data elements
PointList : array(1..n)RoutePoint
AddressList : array(1..n)AddressPoint
Route : geoObject(line)
...
included processing elements
bb-Geocoding
bb-FeatureSelection
mediator behaviour
bb-Routing : { $\surd \rightarrow$ *init*⟨PointList[0..n]; mobType(car,bike,foot) | orig≠dest⟩,
I-init⟨... | PointList↦*oftype*(AddressPoint)⟩ → *check*⟨listOK⟩,
defAOI ∨ *check*⟨... | listOK=*false*⟩ → *I-reject*⟨...⟩,
check⟨...⟩ → *I-inLst*⟨...⟩,
I-inLst⟨...⟩ → *I-OutLst*⟨...⟩,
I-outLst ∨ *check*⟨... | listOK=*true*⟩ → *defAOI*⟨...⟩,
defAOI⟨...⟩ → *I-AofInt*⟨...⟩,
I-AofInt⟨...⟩ → *bb-ABpath.entry[1]*⟨...⟩,
bb-ABpath.entry[1]⟨...⟩ → *pathCalc*⟨...⟩,
pathCalc⟨...⟩⟨... | lastSect=*true*⟩ → *bb-ABpath.exit[1]*⟨...⟩,
bb-ABpath.exit[1]⟨...⟩ → *cartRep*⟨... | numPnt=2⟩,
bb-ABpath.exit[1]⟨...⟩ → *join*⟨...⟩,
join⟨...⟩ → *cartRep*⟨...⟩ }
...
bb-Routing, bb-Geocoding *interact on* I-inLst, I-outLst
bb-Routing, bb-FeatureSelection *interact on* I-AofInt, I-featSet
end;

Chapter 8

Case study: land information service

This chapter presents the case study of the land information service, which we used to illustrate the use of our design methodology. The chapter is structured as follows: section 8.1 introduces of the case, section 8.2 describes informally the behaviour of the service, section 8.3 presents the external perspective model, and section 8.4 presents the internal perspective model.

8.1 Overview

The land information service, LI-service for short, is a service centred on the generation of information about a piece of land. This includes, a.o., the actual use, the ownership, location and size. The main objective of the service is to determine the price and taxes of specific areas of land. The LI-service is the responsibility of the Land Information Agency.

Every taxable piece of land (parcel) is associated with a tax payer, who can be the owner or the tenant of the land. The tenant is the person that is currently exploiting the land. There are multiple factors used in the determination of the tax that is assigned to each parcel, the most important factors being the area of the parcel and its current use. Taxes are to be charged to the tax payers twice a year.

The taxation regulation states that the tax pertaining to a parcel should be calculated as a function of three main aspects: 1) land use, 2) productivity and 3) infrastructure. The land use determines the base tax percentages for a parcel. A parcel may currently have more than one type of use, or may not be used in its totality for production. Productivity is a function of type the land use, area, and production conditions (e.g., temperature, rainfall, etc.). Infrastructure has to do with the location of the field, that is distance to urban centres, road network, etc.

Multiple types of raw data are required for the execution of this service, for instance, timely imagery covering the areas of interest for the taxation period under consideration, geometric features describing parcel boundaries and nearby infrastructure,

weather data, etc. In addition to this raw data, specialised processing tasks are needed to extract from the raw data, the necessary information for the tax calculation. Examples include, a.o., image processing and geo-referencing.

The Land Information Agency does not count with all the data and processing resources necessary to make the service possible. Furthermore, the Land Information Agency neither has the expertise required to implement and maintain the complete service, nor the responsibility for generating and maintaining much of the necessary data. Nonetheless, the Land Information Agency is responsible for the tax calculation, and therefore has to identify a mechanism to realise it. This is achieved by designing a collaborative system to allow the Land Information Agency to make use of specialised services from other geo-service providers whose expertise are in line with the particular needs of the LI-service.

In this specific example, certain data (meteorological data for example) does not have to be produced by the Land Information Agency, but rather obtained from the responsible provider, even though some processing may be necessary to derive the values as required for the tax calculation; and, periodic information about land use requires the exploitation of raw data (satellite images) from which the land use data for the specific period can be extracted, this data can obtained by means of an additional service that complies with the particular requirements for this case.

8.2 Service walkthrough

The LI-service is initiated every time the Land Information Agency receives a request from the government to determine the taxes for a group of taxable parcels. Four major steps can be identified in the execution of the LI-service. The first step encompasses the activities that lead to the obtention of the necessary data to start any processing. These activities focus on: 1) accessing data repositories to obtain, a.o., 1) the geometric features that correspond to the parcels registered as taxable, and 2) up-to-date imagery data corresponding to the areas where the set of taxable parcels are located.

Collected data is then converted into a format that is suitable for the calculation process. This implies, for example, that uncontrolled imagery has to be converted into sufficiently geo-referenced imagery, that is with the positional accuracy and quality needed to be in conformance with the LI-service. This could involve the measurement of control points and ground points if necessary.

The second major step is to utilise the imagery and any other sources to extract the information about current use of the land for the areas of interest. Included here are activities of image classification and change detection to assign to each parcel the current use or uses of the land. This step finishes when all the information on land use for the parcels in the area(s) of interest has been obtained.

The third step involves the collection of information regarding productivity, such as temperature and rainfall, and regarding infrastructure such as, proximity to cities, roads, railways, etc.

In the fourth step, which is the real competence of the Land Information Agency, the results of previous steps (the geometry of parcels, the types of land uses, the weather data, the infrastructure data) are used in the actual calculation of the taxes.

8.3 External perspective model

We start the development by creating a *service definition* from the external perspective. According to the external perspective, we use a single functional entity, LIS (short for Land Information System), to represent the overall system. This entity plays the role of the service provider entity. This entity is associated with a behaviour, which in this case we have called LI-Service.

Figure 8.1 depicts the external perspective model of the Land Information Service. The model was developed based on the description provided in section 8.2. The behaviour block LI-Service is introduced to capture the interaction contributions of the system for this particular service. Two interactions are defined, tpList and tForm.

Interaction *tpList* represents the reception by the Land Information Agency of a taxation request including, a list of tax payers, and the identification numbers of the taxable parcels associated with each tax payer. The list of tax payers corresponds to a specific region whose taxes are calculated during the current period. Interaction *tForm* represents the delivery of the set of invoices one for each of the tax payers listed in the initial request. According to this behaviour definition, interaction tpList is followed by interaction tForm. The textual representation of the behaviour depicted in Figure 8.1 is listed below:

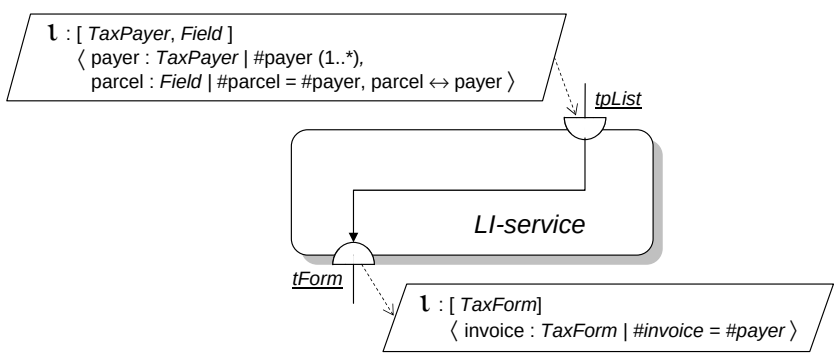


Figure 8.1: The Land Information Service external perspective

$$\begin{aligned} \mathit{LI\text{-}Service} : \{\, \surd \rightarrow \underline{\mathit{tpList}} \langle\, \iota = \mathsf{payer} : \mathit{TaxPayer} \mid \sharp\mathsf{payer}(1..*), \\ \mathsf{parcel} : \mathit{ParcelNo} \mid \sharp\mathsf{invoice} \geq \sharp\mathsf{payer} \,\rangle; \\ \underline{\mathit{tpList}} \rightarrow \underline{\mathit{tForm}} \langle\, \iota = \mathsf{invoice} : \mathit{TaxForm} \mid \sharp\mathsf{invoice} = \sharp\mathsf{payer} \,\rangle \;\; \} \end{aligned}$$

In Figure 8.1, the information attribute of interaction *tpList*, $\iota_{\underline{\mathsf{tpList}}}$, is depicted as an item that contains two lists. The list of tax payers, which is represented by the variable payer of type *TaxPayer*, and the list of associated parcels, which is represented by the variable parcel of type *ParcelNo*. There is a condition on the number of tax payers, represented by the constraint $\sharp\mathsf{payer}(1..*)$, which defines that the list of tax payer should contain at least on member. There is a condition on the list of parcels, represented by the constraint $\sharp\mathsf{parcel} \geq \sharp\mathsf{payer}$, which indicates that the number of parcels should be at least the same as the number of tax payers. There is also a condition on the list of parcels, represented by the constraint $\sharp\mathsf{parcel} \leftrightarrow \sharp\mathsf{payer}$, stating that each parcel should be associated with a tax payer.

The information attribute of interaction *tForm*, $\iota_{\underline{\mathsf{tForm}}}$, is represented with an item that contains a list of invoices. This list is represented by the variable invoice of type *TaxForm*. There is a condition on the number of tax payers, represented by the constraint $\sharp\mathsf{invoice} = \sharp\mathsf{payer}$, which defines that there should be one invoice for each tax payer.

For each of the interactions defined in the *service definition*, one or more information diagrams are developed to capture the information used or established during the execution of the interaction. For example, Figure 8.2a illustrates the information diagram developed for the item associated with the interaction *tpList*. Interaction *tpList* requires an input, TaxRequest, that represents the information used. TaxRequest is composed of a two sets of information, Payer and Parcel. This information represents the details of the parcels to be taxed and their associated tax payers. Similarly, Figure 8.2b illustrates the information diagram for interaction *tpForm*.

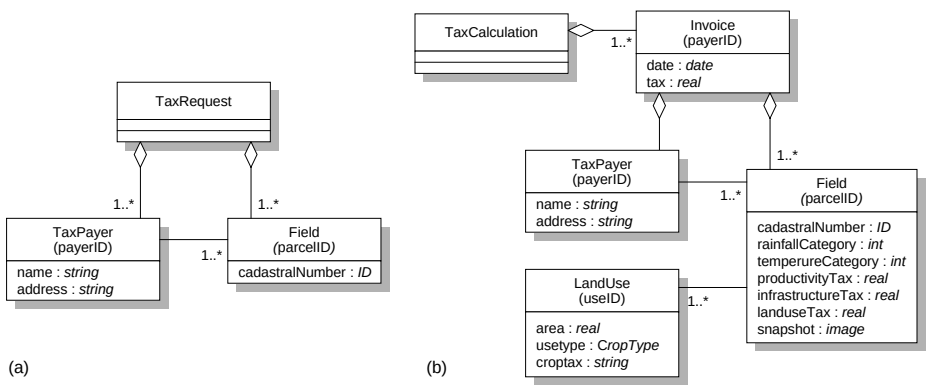


Figure 8.2: Information diagram for the interactions *tpList* (a) and *tForm* (b)

8.4 Internal perspective

The internal perspective is developed in three steps. In the first step, we refine the *service definition* and identify a number of architectural elements. In the second step we create a behaviour specification that defines the relations between these architectural elements and the corresponding constraints. In the third step, we specify the internal behaviour of the architectural elements.

8.4.1 Introduction of internal actions

Our methodology prescribes a decomposition based on functionality in order to identify the services required to assemble the LI-service. Using the behaviour walkthrough presented in section 8.2, we identify the following set of required services.

1. A **Parcel Feature Service** is used to obtain boundaries of parcels. The CadastralNumbers of the parcels is provided as input to the service and the result is a theme that contains the polygons that define the shape and location of the parcels.
2. An **Imagery Service** uses the coordinates from the geometric extent (limits) of the parcel's theme, and the dates of the taxable period to obtain the satellite images covering the areas where the parcels are located. These images correspond to the starting and ending dates of the taxable period.
3. An **Image Processing Service** is used to geo-reference the satellite images and obtain a mosaic of the area of interest. The input to this service is the set of satellite images, and the result is the geo-referenced image mosaic.
4. A **Change Detection Service** compares images to identify the areas within the parcels that have had a change in the landuse.
5. An **Feature Extraction Service** is used to define the different types of landuse within the taxable parcels.
6. A **Weather Data Service** is used to obtain the information regarding temperature, rainfall, etc. of the area of interest during the taxable period.
7. A **Feature Selection Service** is used to obtain information about nearby infrastructure.
8. A **Taxation Service** is used to determine each parcel's tax and deliver the invoices.

Figure 8.3 depicts the general process flow for the LI-Service based on the eight distinctive independent services identified above. This process flow represents a set of steps to realise the service. Optimality in this sense this process represents the best

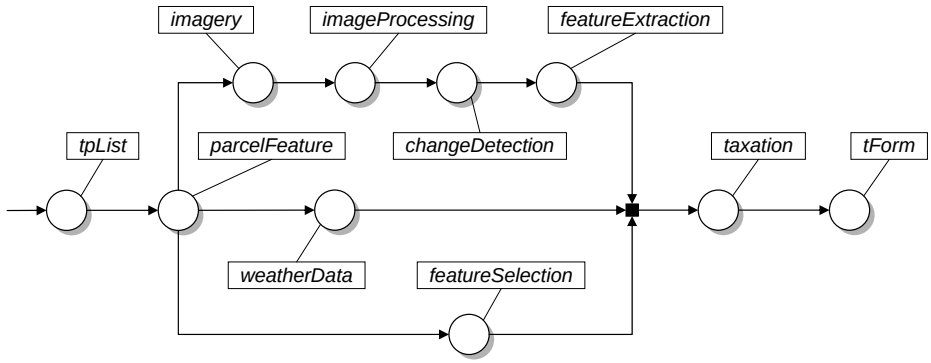


Figure 8.3: General service process

approach to solve the problem given the current status of the system. The objective here is to identify independent tasks that can be executed separately, if possible in parallel and which functionality can be obtained from different resources (architectural elements) within the infrastructure. That is, since services are in principle independent of each other, they may exist within one organisation or they may be supplied by an external service provider. They are specified in a way such that they can be used in a service realisation in spite of their physical location or specific implementation The identification and definition of these independent services facilitates the sharing of services and the reuse of functionality.

8.4.2 Restructuring

With the general process flow described we can proceed to the definition of the mediator element. For this purpose we discriminate all independent elements and then we define the relations between these elements and the mediator, such that the LI-service can be realised. In this case the mediator is called service-handler. This element is responsible not only for the interface to the client application, and for hiding the existence of the other elements from the user, but it also plays the connector role to connect and control the relations between the other elements.

Figure 8.4 shows a behaviour definition including all of the elements identified before and their corresponding interconnections. At this level of abstraction we depict the relations between elements as single interactions. These interactions need to be refined in order to fully specify the relations they represent.

Figure 8.5 shows the refined definition of the interaction between the elements weatherData and serviceHandler. The Weather Data Service is an independent service that provides weather information for specific periods of time. The service requires as inputs two dates and an area of interest. These inputs are modeled in Figure 8.5 as startDate, endDate, and region respectively. The service also allows the provision of parameters to specify which type of weather data is being requested. If no parameter

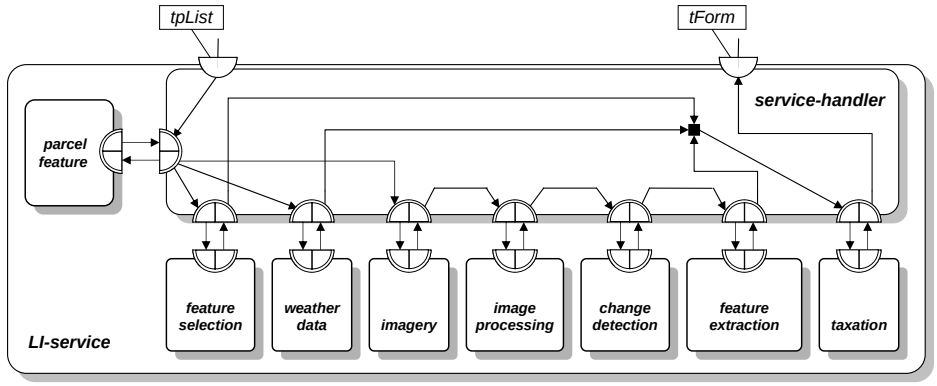


Figure 8.4: Initial internal perspective design

is provided the Weather Data Service generates only temperature data. The default option of the parameter, temperature, is shown underlined.

The serviceHandler element interacts with the weatherData element at interactions *wdReq* and *wdResp*. The serviceHandler element asks the weatherData element for the necessary weather data through the interaction *wdReq*. The information attribute of interaction *wdReq.serviceHandler* consists of three elements, startDate, endDate, and region, and the list of parameters, e.g., temperature and rainfall. These are used by the Weather Data Service to retrieve the corresponding weather data. The interaction *wdReq* is followed by the interaction *wdResp*, which delivers the requested data.

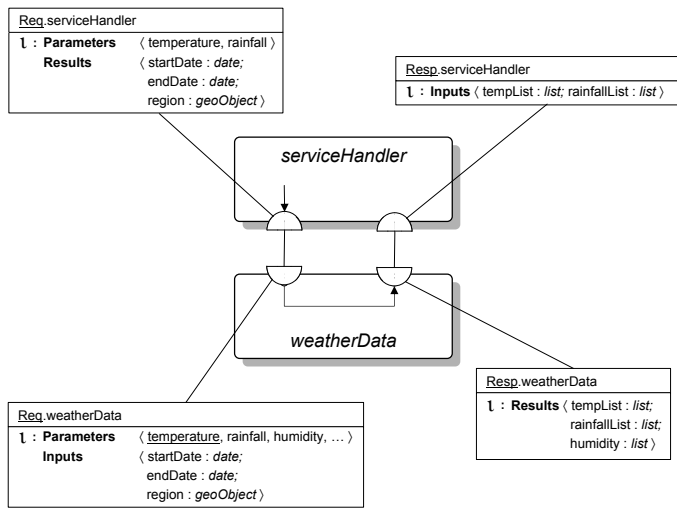


Figure 8.5: WeatherData service interaction diagram

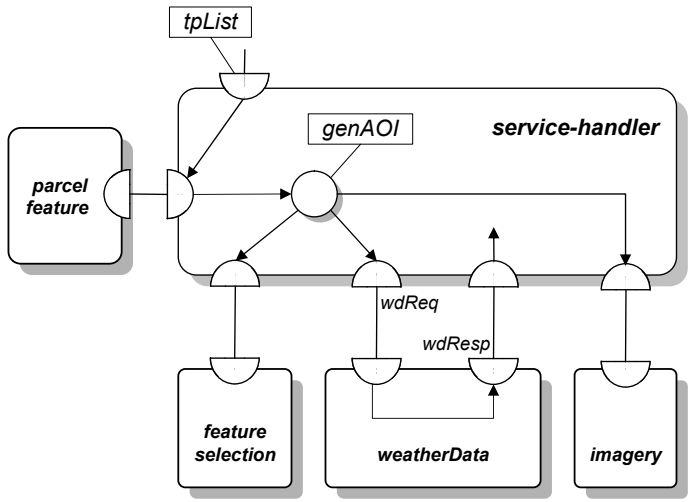


Figure 8.6: Introduction of the genAOI action

At the start of the LI-service the information corresponding to the area of interest is not known. The only information known are the identifiers of the parcels to be taxed. Therefore, before interaction *wdReq.serviceHandler* can take place it is necessary to determine this area of interest such that the region element of the information attribute of *wdReq.serviceHandler* can be provided.

To determine the area of interest we have to obtain the polygons corresponding to the parcels being taxed. These polygons are obtained through the Parcel Feature Service. To define the area of interest we introduce the action *genAOI* (see Figure 8.6).

The process of refining the interactions to define interaction contributions and inserting actions to carry out any additional task needed to enable the connections between two elements is repeated for all the other interactions defined for the service. In the diagram of Figure 8.6, the relationships between interactions are refined through the insertion of actions. Nevertheless, the ordering in which interactions should be executed is preserved. For simplicity reasons we have only shown the refinement of the interaction between the serviceHandler and the weatherData elements.

8.4.3 Assignment of sub-behaviours

In this example we assume that the services provided by the ImageProcessing element and the FeatureExtratction element are unavailable. This in order to illustrate the iterative nature of the methodology. We refined imageProcessing behaviour block by inserting a number of relevant actions that are executed by the system as a result of the occurrence of the initial interaction of Image Processing Service, which is called *imageList*.

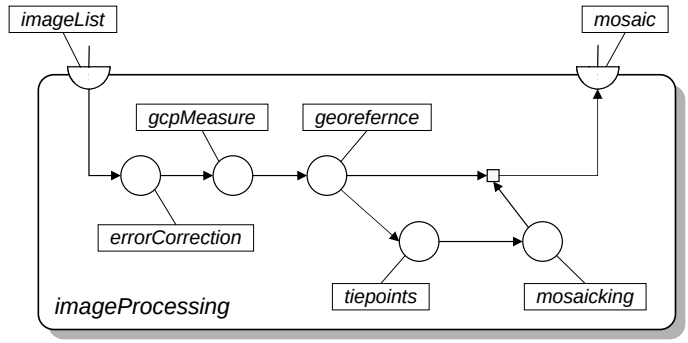


Figure 8.7: Behaviour definition for the imageProcessing element

The Image processing service involves the activities that lead from uncontrolled imagery to sufficiently georeferenced imagery. The activities are: *errorCorrection* for specifying the correction models for camera motion and lens errors, *gcpMeasure* for the observation and measurement of control points and ground points, establishing a sufficiently rich bundle of equations for adjustment, *georeference*, which involves the actual adjustment of the images, *tiepoints* for the determination of common points between adjacent images, and *mosaicking* which comprises the joining of the various images. The last two actions, *tiepoints* and *mosaicking* only take place when there is more than one image. Figure 8.7 depicts the behaviour definition of the Image Processing Service.

Figure 8.8 shows the behaviour definition for the Feature Extraction Service. This behaviour is initiated with the interaction *feReq*. If the request includes aerial photographs, the interaction *feReq* is followed by the action *stereoView*. Otherwise, the *featureOverlay* action is performed. Action *stereoView* is followed by action *featureOverlay*. Action *stereoView* represents the the creation of a stereoscopic view of the area of interest. Action *featureOverlay* represents the overlaying of vector features (parcel boundaries for the case of the LI-service) to determine the areas from which information should be extracted. Action *featureOverlay* is followed by actions *geometry*,

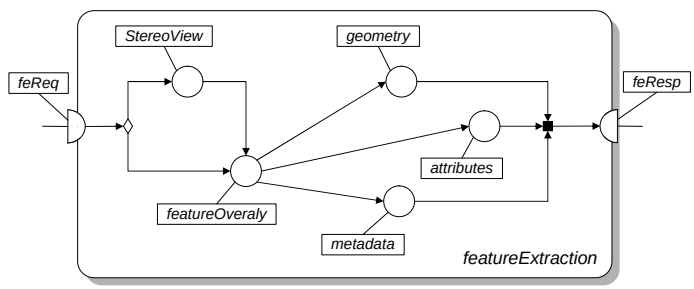


Figure 8.8: Behaviour definition for the featureExtraction element

attributes and *metadata*. These activities represent the discovery of features, the extraction of their geometry and attributes, and the generation of the associated metadata according to the extraction specifications. These last three actions are followed by interaction *feResp* that represents the end of the process and the delivery of the extracted information.

Figures 8.7 and 8.8 represent the refinements of the imageProcessing and featureExtraction behaviours respectively. We generated this refinement in one single step, mainly because of the relatively simple nature of the services. For the case of a more elaborated service this refinement can be generated in a series of consecutive refinement steps.

Chapter 9

Conclusions

This chapter presents the most important conclusions reached throughout our research. The chapter also summarises the main contributions of the work, and outlines some directions for further research.

9.1 General considerations

Making geo-information services available as flexible distributed applications complementary to exiting geographic data sharing facilities has lately drawn the attention of geo-information scientists. Designing this type of systems presents various challenges due to the complex nature of their requirements and internal structures.

Some of the main concerns around the design of these systems are, a.o.: how to specify and describe their overall functionality; how to specify and describe the functionality of these systems internal components; how to facilitate reuse and customisation of these systems.

In this thesis we show that a model-driven design approach can provide an adequate way of coping with these issues. We explain how models can be used as a means to design the system, and to properly describe its functionality to facilitate reuse.

We emphasise that, in order to use models efficiently in the design of geo-information systems, these models should capture the correct and necessary detail to describe the system. We achieve this with the use of a structured methodology to guide the development of these models.

We propose a methodology for the architectural design of geo-information systems that is organised around system perspectives. These perspectives allow the separation of different concerns throughout the design process and the stepwise development of a number of related system specifications.

The methodology comprises a set of design concepts and specialisations thereof, to

be used in the creation of models. We identify a set of architectural elements to represent the different types of components of geo-information systems. We also defined an architectural style and a development trajectory that facilitate identification of architectural elements, and the definition of services as combinations of these elements.

Our methodology also makes use of models to facilitate the sharing of functionality among multiple systems and the design of elaborated services by reusing this functionality. This capability resides in the incorporation of a repository that allows to organise the creation, updating, validation, accessing and sharing of service models and service instances. This allows the arrangement of elementary services into elaborated combinations for richer functionality capable of providing specialised and more client specific services.

9.2 Main contributions

The process of designing and creating geo-information systems that support cooperative work among multiple geo-service providers has, up to now, been done without the support and guidance of an adequate methodology. A drawback of the approaches used by geo-information designers is that they mainly focus on the informational aspects of the system and they have limited mechanisms for expressing or specifying behaviour. This can be explained by the complexity of the data structures needed to manipulate geographic data, therefore great emphasis was placed on this aspect.

We have developed a methodology that enables the modelling of both behavioural and informational aspects of geo-information systems in an integrated way. To derive the methodology we first identified the specific characteristics and nature of geo-information systems and studied the needs of distributed geo-information processing.

Based on this study we introduced the concept the Geo-information Service Infrastructure (GSI). The GSI concept clearly highlights the characteristics and needs of modern geo-information processing. The GSI builds on the existing principles for data sharing of the Geo-information Infrastructure concept. We use the term GSI to refer to a type of geo-information provision system from which specialised information products and services can be obtained by exploiting the artefacts of a set of collaborating geo-service providers. We also present a supporting architecture for the deployment of GSI services.

Conventional geo-information systems are commonly bound to the provision of data as their only service. The exploitable artefacts of a GSI include, in addition to data, operations, processes, value-added products and resources. These are referred to as elementary services. Within the GSI a common method is used to describe these elementary services and their interfaces, such that they can be accessed, combined and managed to create compound services. These services are defined to handle more elaborated and specialised geo-processing tasks.

To support the design of these type of system and more specifically of the services they provide, we have developed GSDM, a methodology to guide theses systems design process. GSDM proposes the design of GSI services according to different levels of concern. Concern levels are organised into two system perspectives, the internal perspective and the internal perspective. Different specifications of the system can be developed according to a stepwise approach based on these interrelated perspectives.

The external perspective aims at identifying and explicitly delimiting the scope of the system under development and helps determining the objectives of the development process. The internal perspective aims at describing the internal system structure in terms of compositions of simpler or more elementary architectural elements. Arrangements of these architectural elements form so-called service specifications. A service specification identifies a group of architectural elements and describes how these elements interact to provide a required service.

GSDM distinguishes three types of architectural elements, viz. data elements, processing elements and connecting elements. The *data elements* represent the information that is used, manipulated and/or generated by the system. The *process elements* represent the geo-processing capabilities of the system, which can perform transformations on data elements. The *connecting elements* or *mediators* coordinate the interactions between the other architectural elements, and provide an interface to the service user.

Our methodology uses the so-called mediator pattern to structure compositions of architectural elements. This results in the organisation of a set of services into a behaviour definition that has a single coordinating element. Among other benefits, this approach makes the service realisation accountable for the user, and facilitates the use of workflow languages to implement the mediator behaviour, which choreographs the use of third-party services.

The use of perspectives, stepwise refinement and the mediator pattern allows for a better control over the design process of a GSI system, such that the designer can maintain the integrity and conformance of the design itself with respect to the successive transformations that an abstract design must undergo, until a concrete design suitable for realisation is obtained.

We introduce general design concepts, based on ISDL, to represent the different architectural elements. Based on these concepts, specification techniques and guidelines are presented. These specification techniques are targeted to the characteristics of the particular concerns addressed in each perspective. An architectural style to guide the refinement process is also presented based on the design concepts and specification techniques.

At the centre of a GSI system lies the repository service. This repository allows to organise the creation, updating, validation, accessing and sharing of service models and service instances. We emphasise the use of models as the mechanism to disclose information about services and to design more elaborated services out of combinations

of existing elementary services. For this approach to work, models need not only to be interchanged between participants, but they also have to be understood by all parties involved. This is achieved by defining a metamodel on which all service models are based. We introduce a metamodel that provides a rigourous abstract syntax for defining models. The metamodel is used here to define a set of design concepts and their relationships, which one can use to produce models according to the GSI specific objectives.

The ideas introduced in this thesis provide a reference point for future attempts to establish geo-information infrastructures at local, regional or national level. Data producers and geo-service providers count with an approach and a set of guidelines to help them establish a collaborative infrastructure, which they can use to generate value-added services. Those who have information infrastructures already in operation can make use of these ideas to envision the next evolutionary steps for their infrastructures. Geo-service providers can use the method and concepts presented in this work to specify and share each others functions and turn them into combined, distributed, collaborative services.

9.3 Further Research

The work discussed in this thesis represents the initial stages on the establishment of a much needed methodology for the design of distributed geo-information services, and therefore, it can be continued in a number of ways. We identify a number issues that require further investigation: the expansion of the methodology scope, the incorporation of quality of service criteria, the provision of automated support for the success of large scale development projects, the study of the relationships between our methodology and the OGC reference model [OGC03b].

The scope of the methodology presented in this work concentrates on the design of geo-information systems, at the so called internal and external perspectives. These two perspectives address the service specification and design phases of the development process. An additional contribution could be to investigate and develop harmonious design trajectories and views for the business or enterprise perspective, and for the deployment and management perspectives.

Another step forward is to identify appropriate design concepts to specify quality of service. We have proposed a method to enable the design of specialised tasks that are achieved by combining elementary services that together provide a desired functionality. Since there are multiple possible configurations of elements that could comply to the same requirements, it is still necessary to identify which quality parameters should be embedded in the designs, such that it is possible to determine the suitability of different designs against specific project constraints, like time or cost.

In the same way one could further investigate the use workflow management systems to choreograph the execution of actual service instances. By mapping the designs

obtained with our methodology to an adequate workflow model it would be possible to optimise the use of resources, and to manage effectively the collaboration between members of a service infrastructure.

Automated support for the execution of large development projects, such as a fully-fledge GSI system are another area of interest. Tools to automate validation of and conformance assessment of specifications are being currently developed. Some other tools could be developed to manage the service repositories that support a GSI, for instance, for version control.

An interesting use of automation lies in the transformation of models. We target our methodology to facilitate the integration of distributed geo-services via the development of general models across various phases of the system lifecycle. These models are system 'or platform' independent. The transformation and specialisation of these models into platform specific would improve the capability of our methodology increasing specially the potential of the service repository. In this regard one can investigate the relationship of our methodology with the Model Driven Architecture [OMG01b] initiative of the Object Management Group (OMG).

Recently, the OpenGIS Consortium (OGC) [Ope04] has released a framework known as the OGC Reference Model [OGC03b]. The main goal of this framework is to support the specification and implementation of interoperable solutions and applications for geospatial services, data, and applications. To a certain extent, the idea behind our methodology is similar to the OGC Refrence Model. In both cases an attempt is made to capture system information before the specification of the system itself, although in our case we partially address issues of the business domain and we do not address the technology domain at all. Further, both aim at producing platform-independent specifications that can be later specialised accordingly. Finally, both rely on the use of models as a mechanism to disclose information on the system and to facilitate the chaining of services. Still, additional investigation is needed to combine the best of both approaches or to define how they can influence each other.

Appendix A

Services metadata

You will never be able to discover new oceans
unless you have the courage to lose sight of the shore.

Hannah Whitall Smith

A.1 Service descriptions

A service represents the contribution of a system or part thereof to its surrounding environment that is of value to the end-user. In our case, this contribution corresponds to some functionality that is provided by a geo-information system over a network.

These services, which range in size from small low-level services that represent single units of functionality, to high-level services that fuse lower-level services for processing to provide more elaborated functions to users, encompass a serie of networked resources.

To facilitate the search and discovery of these services, an adequate description in the form standardised descriptive metadata has to be associated with them (see section 7.4). These service descriptions should be reach enough to convey enough information, which clients can use to discover and utilise such services. Such representations should be both human and machine readable, to not only facilitate discovery and use but also to ease their integration in elaborated end-user applications.

In our case this metadata encompasses a set of metadata elements based on the Dublin Core Metadata Initiative (DCMI) [DCM03a].

The Dublin Core metadata standard is a simple yet effective element set for describing a wide range of networked resources. The Dublin Core metadata element set is a standard for cross-domain information resource description. The DCMI defines an information resource to be "anything that has identity", there are no fundamental restrictions to the types of resources to which Dublin Core metadata can be assigned.

A.2 The metadata elements

In this section we introduce and explain the set of metadata elements for the general description of GSI services. Specific metadata elements, such as those which are only related to data services for example, are not listed here.

This metadata deals only with descriptive information about a service, the actual service definitions are store in the repository (see appendix B).

Element Name : Title
Definition : A name given to the resource.
Comment : The element provides a name by which the resource is formally known and could be identified, e.g., "Movement tracking, guidance and control", "Landsat TM datasets of Overijssel, The Netherlands"

Element Name : Creator
Definition : The entity primarily responsible for making the content of the resource
Comment : This element should be used to indicate the person or organisation that put the service together, e.g., "Alvarez Casallas, Luisa Liliana", "Institute for Environmental studies", "Dutch Topographic Service"

Element Name : Date
Definition : A date of an event in the lifecycle of the resource.
Comment : Typically this element is associated with the creation or availability of the resource.

Element Name : Identifier
Definition : An unambiguous reference to the resource within a given context.
Comment : Recommended best practice is to identify the resource by means of a string or number conforming to a formal identification system.

Element Name : Description
Definition : An account of the content of the resource
Comment : Some sort of explanation of what the resource does, or what it is useful for.

Element Name : Format
Definition : The physical or digital manifestation of the resource
Comment : Typically, Format may include the media-type or dimensions of the resource. Format may be used to identify the software, hardware, or other equipment needed to display or operate the resource. Examples of dimensions include size and duration, e.g., "image/tiff 12MB".

Element Name : COVERAGE
Definition : The extent or scope of the content of the resource.
Comment : Typically, Coverage will include spatial location (a place name or geographic coordinates), temporal period (a period label, date, or date range) or jurisdiction (such as a named administrative entity), for example, Thesaurus of Geographic Names of Colombia [TGNC]). Named places or time periods should be used in preference to numeric identifiers such as sets of coordinates or date ranges.

Element Name : CATEGORY
Definition : A topic of the content of the resource.
Comment : Subject is expressed as keywords, key phrases or classification codes that describe a topic of the resource. Recommended best practice is to select a value from a controlled vocabulary or formal classification scheme, e.g. "Visualisation service",

Element Name : RIGHTS
Definition : Information about rights held in and over the resource.
Comment : Rights information often encompasses Intellectual Property Rights, Copyright, and various other Property Rights. If the Rights element is absent, no assumptions may be made about any rights held in or over the resource.

Element Name : LANGUAGE
Definition : A language of the intellectual content of the resource.
Comment : Recommended best practice is to use RFC3066 which, in conjunction with ISO639 [ISO03] define two and three primary language tags with optional subtags. Examples include "en" or "eng" for English, "akk" for Akkadian", and "en-GB" for English used in the United Kingdom.

A.3 Service metadata schema

Existing approaches to the implementation of service and resource descriptions over a network include, a.o., the Web Service Description Language (WSDL) [W3C01a], the Universal Description Discovery and Integration (UDDI) standard service registry [BCE+02], the Dublin Core Metadata Initiative (DCMI) [DCM03a], the service representation approach to support agent communication (AIGA) [Nol03, AIG03], the Semantic Web [BLHL01, DCM03b], the Darpa Agent Markup Language for Services (DAML-S) [DHM+01, DAM03], the Metadirectory Service (MDS) for Grid computing [CFFK01]. All of these approaches define different mechanisms to encode information about network resources.

We have selected the Resource Description Framework (RDF) [W3C99] for the representation of service descriptions, and the eXtensible Markup Language (XML)

[W3C01b] for the encoding these descriptions. The Resource Description Framework (RDF) and the eXtensible Markup Language (XML) are both World Wide Web Consortium (W3C) [W3C03] recommendations for sharing information over the web.

The schema presented below includes the elements that we have defined as metadata. The schema also includes elements to encode specific information about the service that are based on the definition of the service repository and its associated metamodel (see section 4.4). These later elements include, a.o., inputs, parameters, results, their corresponding data types.

For each GSI service the above information stored in XML and made available across the network. When a user needs to locate a service providing a particular type of functionality, it can search the metadata trough traditional means using any XML enable search engine. The service metadata schema is listed below. To identify how a particular service provider realise a service, a user can search through the information provided in XML containing the actual service specification (see Appendix B).

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ============================== -->
<!--     GSI Services Metadata      -->
<!-- ============================== -->
<!--    International Institute     -->
<!--  for Geo-Information Science   -->
<!--             and                -->
<!--    Earth Observation (ITC)     -->
<!-- ============================== -->
<!--   University of Twente (UT)    -->
<!--     Centre for Telematics      -->
<!--  and Information Technology    -->
<!-- ============================== -->
<!--   Enschede, The Netherlands    -->
<!-- ============================== -->
<!-- ============jmmg============== -->
<!-- ============================== -->

<rdf:RDF xml:lang="en"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303"
    xmlns:wdxml="http://www.w3.org/TR/2000/WD-xmlschema-2-20000407"
    xmlns:dc="http://purl.org/dc"
    xmlns:gsi="http://www.itc.nl/morales/gsi/schemas">

  <rdf:Description about="">
    <dc:Title>Description for GSI Services definitions</dc:Title>
    <dc:Creator>Javier Morales</dc:Creator>
    <dc:Date>2002-11-14</dc:Date>
    <dc:Format>text/xml</dc:Format>
    <dc:Description>Description for GSI Services definitions</dc:Description>
    <dc:Subject>Geo-information services</dc:Subject>
  </rdf:Description>

  <!-- ============================ -->
  <!-- General service information -->
  <!-- ============================ -->
```

```
<rdf:Description ID="Service">
  <rdf:type resource="rdfs:#Class"/>
  <rdfs:subClassOf rdf:resource="rdfs:#Resource"/>
  <rdfs:comment>GSI services main abstract class</rdfs:comment>
</rdf:Description>

<rdf:Description ID="name">
  <rdf:type resource="rdfs:#Property"/>
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="wdxml#string"/>
  <rdfs:comment>The name of the service</rdfs:comment>
</rdf:Description>

<rdf:Description ID="creator">
  <rdf:type resource="rdfs:#Property"/>
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="wdxml#string"/>
  <rdfs:comment>The creator (e.g., company, person) of the service</rdfs:comment>
</rdf:Description>

<rdf:Description ID="version">
  <rdf:type resource="rdfs:#Property"/>
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="wdxml#string"/>
  <rdfs:comment>The version of the service</rdfs:comment>
</rdf:Description>

<rdf:Description ID="description">
  <rdf:type resource="rdfs:#Property"/>
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="wdxml#string"/>
  <rdfs:comment>A human readable description of the service.</rdfs:comment>
</rdf:Description>

<rdf:Description ID="category">
  <rdf:type resource="rdfs:#Property"/>
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="wdxml#string"/>
  <rdfs:comment>A class representing geospatial processing functions.</rdfs:comment>
</rdf:Description>

<rdf:Description ID="documentation">
  <rdf:type resource="rdfs:#Property"/>
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="wdxml#string"/>
  <rdfs:comment>A URL to documentation on the service.</rdfs:comment>
</rdf:Description>

<rdf:Description ID="inputDataType">
  <rdf:type resource="rdfs:#Property"/>
  <rdfs:domain rdf:resource="#DataType"/>
  <rdfs:comment>The input data type that the service requires.</rdfs:comment>
</rdf:Description>

<rdf:Description ID="resultDataType">
  <rdf:type resource="rdfs:#Property"/>
  <rdfs:domain rdf:resource="#DataType"/>
```

```
  <rdfs:comment>The result data type that the service produces.</rdfs:comment>
</rdf:Description>

<rdf:Description ID="numInputs">
  <rdf:type resource="rdfs:#Property"/>
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="wdxml#integer"/>
  <rdfs:comment>The number of inputs that the service requires.</rdfs:comment>
</rdf:Description>

<rdf:Description ID="numResults">
  <rdf:type resource="rdfs:#Property"/>
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="wdxml#integer"/>
  <rdfs:comment>The number of results produced by the service.</rdfs:comment>
</rdf:Description>

<rdf:Description ID="parameter">
  <rdf:type resource="rdfs:#Property"/>
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="#Parameter"/>
  <rdfs:comment>A parameter used by the service.</rdfs:comment>
</rdf:Description>

<rdf:Description ID="numParameters">
  <rdf:type resource="rdfs:#Property"/>
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="wdxml#integer"/>
  <rdfs:comment>The number of parameters required by the service.</rdfs:comment>
</rdf:Description>

<rdf:Description ID="dependantUpon">
  <rdf:type resource="rdfs:#Property"/>
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:comment>Any dependancy that the service needs
                prior to processing.</rdfs:comment>
</rdf:Description>

<rdf:Description ID="numItems">
  <rdf:type resource="rdfs:#Property"/>
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="wdxml#integer"/>
  <rdfs:comment>The number of items that the service requires.</rdfs:comment>
</rdf:Description>

<rdf:Description ID="item">
  <rdf:type resource="rdfs:#Property"/>
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:comment>The items manipulated by the service.</rdfs:comment>
</rdf:Description>

<rdf:Description ID="GeoOperation">
  <rdf:type resource="rdfs:#Class"/>
  <rdfs:subClassOf rdf:resource="#Service"/>
  <rdfs:comment>A class representing geo-spatial
                processing functions.</rdfs:comment>
</rdf:Description>
```

```
<!-- ==================== -->
<!-- Data type information -->
<!-- ==================== -->

<rdf:Description ID="DataType">
  <rdf:type resource="rdfs:#Class"/>
  <rdfs:comment>An abstract class for data</rdfs:comment>
</rdf:Description>

<rdf:Description ID="ImageData">
  <rdf:type resource="rdfs:#Class"/>
  <rdfs:subClassOf rdf:resource="#DataType"/>
  <rdfs:comment>A class representing image data</rdfs:comment>
</rdf:Description>

<rdf:Description ID="geoFeature">
  <rdf:type resource="rdfs:#Class"/>
  <rdfs:subClassOf rdf:resource="#DataType"/>
  <rdfs:comment>A class representing geospatial data</rdfs:comment>
</rdf:Description>

<rdf:Description ID="geoObject">
  <rdf:type resource="rdfs:#Class"/>
  <rdfs:subClassOf rdf:resource="#GeoFeature"/>
  <rdfs:comment>A class representing geographic objects</rdfs:comment>
</rdf:Description>

<rdf:Description ID="Theme">
  <rdf:type resource="rdfs:#Class"/>
  <rdfs:subClassOf rdf:resource="#GeoFeature"/>
  <rdfs:comment>A class representing geographic objects
                of the same type</rdfs:comment>
</rdf:Description>

<rdf:Description ID="Composite">
  <rdf:type resource="rdfs:#Class"/>
  <rdfs:subClassOf rdf:resource="#GeoFeature"/>
  <rdfs:comment>A class representing gropus of geographic objects
                of any type</rdfs:comment>
</rdf:Description>

<rdf:Description ID="Collection">
  <rdf:type resource="rdfs:#Class"/>
  <rdfs:subClassOf rdf:resource="#GeoFeature"/>
  <rdfs:comment>A class representing groups of geographic themes</rdfs:comment>
</rdf:Description>

<rdf:Description ID="Integer">
  <rdf:type resource="rdfs:#Class"/>
  <rdfs:subClassOf rdf:resource="#DataType"/>
  <rdfs:range rdf:resource="wdxml#integer"/>
  <rdfs:comment>The integer value</rdfs:comment>
</rdf:Description>

<rdf:Description ID="Float">
  <rdf:type resource="rdfs:#Class"/>
  <rdfs:subClassOf rdf:resource="#DataType"/>
  <rdfs:range rdf:resource="wdxml#float"/>
```

```
    <rdfs:comment>The float value</rdfs:comment>
  </rdf:Description>

  <rdf:Description ID="Double">
    <rdf:type resource="rdfs:#Class"/>
    <rdfs:subClassOf rdf:resource="#DataType"/>
    <rdfs:range rdf:resource="wdxml#double"/>
    <rdfs:comment>The double value</rdfs:comment>
  </rdf:Description>

  <rdf:Description ID="Short">
    <rdf:type resource="rdfs:#Class"/>
    <rdfs:subClassOf rdf:resource="#DataType"/>
    <rdfs:range rdf:resource="wdxml#short"/>
    <rdfs:comment>The short value</rdfs:comment>
  </rdf:Description>

  <rdf:Description ID="Long">
    <rdf:type resource="rdfs:#Class"/>
    <rdfs:subClassOf rdf:resource="#DataType"/>
    <rdfs:range rdf:resource="wdxml#long"/>
    <rdfs:comment>The long value</rdfs:comment>
  </rdf:Description>

  <rdf:Description ID="Byte">
    <rdf:type resource="rdfs:#Class"/>
    <rdfs:subClassOf rdf:resource="#DataType"/>
    <rdfs:range rdf:resource="wdxml#byte"/>
    <rdfs:comment>The byte value</rdfs:comment>
  </rdf:Description>

  <rdf:Description ID="Boolean">
    <rdf:type resource="rdfs:#Class"/>
    <rdfs:subClassOf rdf:resource="#DataType"/>
    <rdfs:range rdf:resource="wdxml#boolean"/>
    <rdfs:comment>The boolean value</rdfs:comment>
  </rdf:Description>

  <rdf:Description ID="String">
    <rdf:type resource="rdfs:#Class"/>
    <rdfs:subClassOf rdf:resource="#DataType"/>
    <rdfs:range rdf:resource="wdxml#string"/>
    <rdfs:comment>The short value</rdfs:comment>
  </rdf:Description>

  <!-- ==================== -->
  <!-- Parameter information -->
  <!-- ==================== -->

  <rdf:Description ID="Parameter">
    <rdf:type resource="rdfs:#Class"/>
    <rdfs:subClassOf rdf:resource="rdfs:#Resource"/>
    <rdfs:comment>A class representing input parameter necessary to configure
                  the service.</rdfs:comment>
  </rdf:Description>

</rdf:RDF>
```

Appendix *B*

Repository schema

In the same way as the with service metadata, we have selected the Resource Description Framework (RDF) and the eX- tensible Markup Language (XML) to encode service definitions.

Service definitions encompass the behavioural and structural vies of a service as defined in their corresponding internal and external perspective models. The GML (see Appendix C) is used for the encoding of any spatial features or spatial feature definitions associated with a service. Table B.1 shows the structure of an XML document that depicts a service definition.

Table B.1: Structure of the GSDM repository documents

Section	**Description**
Metadata	Information about the service including: the creator, date of creation, general description, etc. (see section A.3)
Member elements	All processing and data elements that form part of the service, if any.
Behavioural description	The behavioural definition, which corresponds to the internal perspective of the service. This section describes how the various associated elements and internal operations (proper to the particular service) are assembled and cooperate to realise the corresponding service.

The *metadata* section represents metadata about the GSI service that can be if use for determining what the service does, who owns the service, when it was created, when it was last modified, etc. (see section A.3). The metadata section also represents the inputs and the results obtained from the service.

The *architectural elements* section represent the necessary processing and data elements that form part of the current service and that are necessary to realise the service function. This is essentially a listing of the elements required and not the order in which they are used or associated.

The *behavioural description* represents the behaviour definition corresponding to the internal perspective model of the service. It includes the exact arrangement of architectural elements including relationships and constraints. This is considered the best approach that a service provider can offer to solve a problem.

Figures B.1, B.2 and B.3 depict the graphical representation of the repository schema. The complete XML repository schema is listed after the figures. The schema conforms to the metamodel for the GSI repository that was introduced in section 4.4.

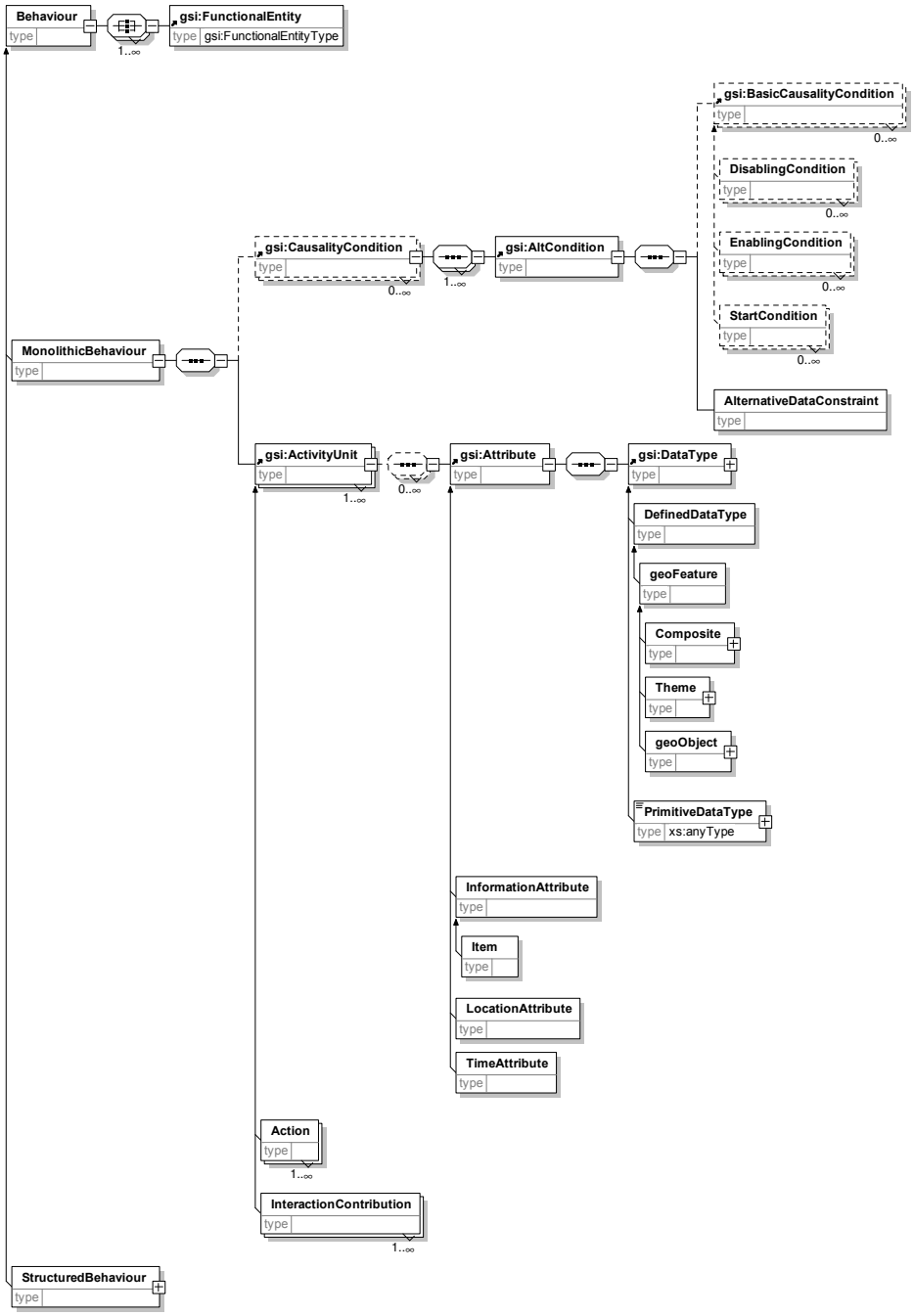


Figure B.1: Repository schema - part I

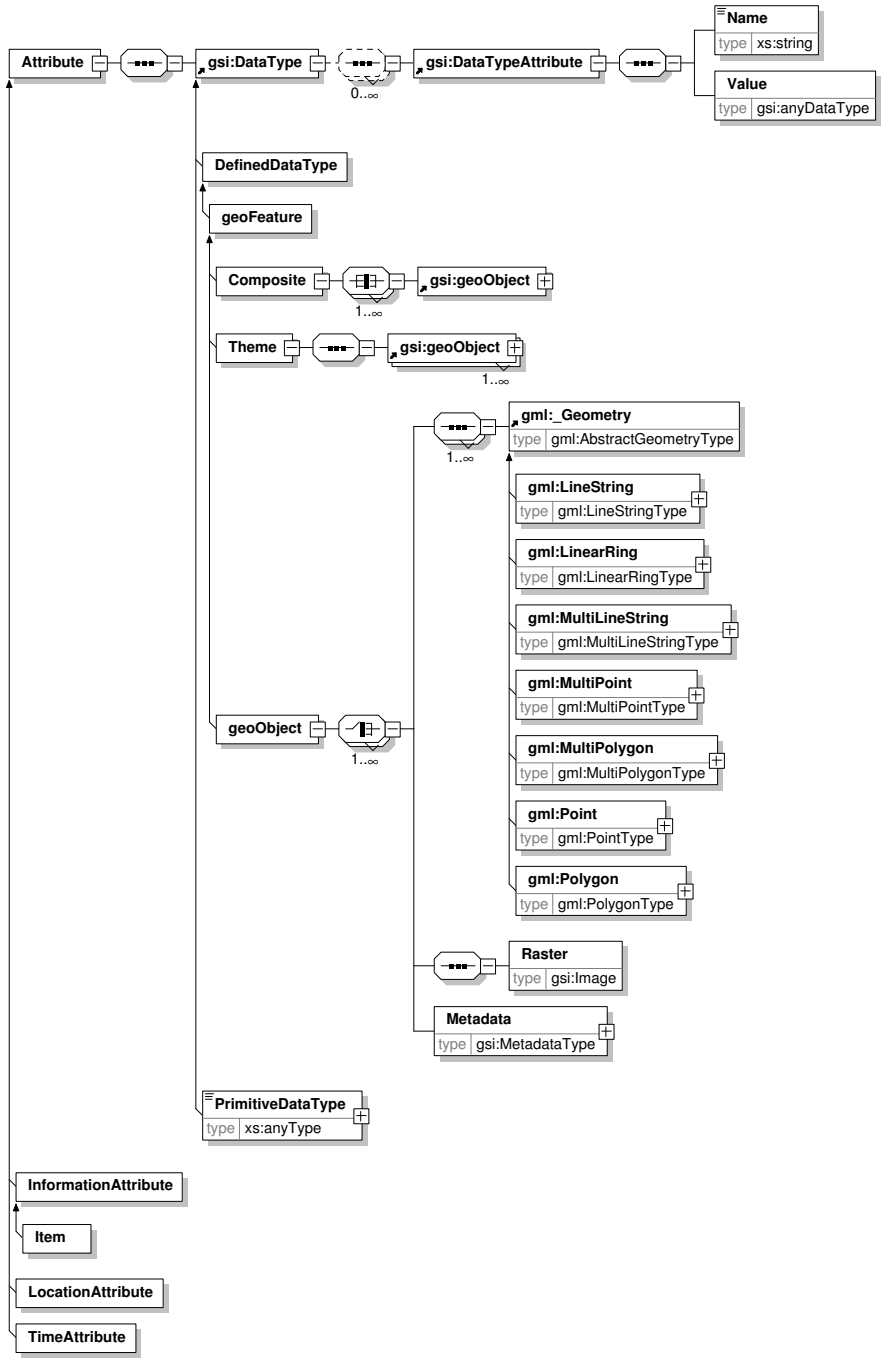


Figure B.2: Repository schema - part II

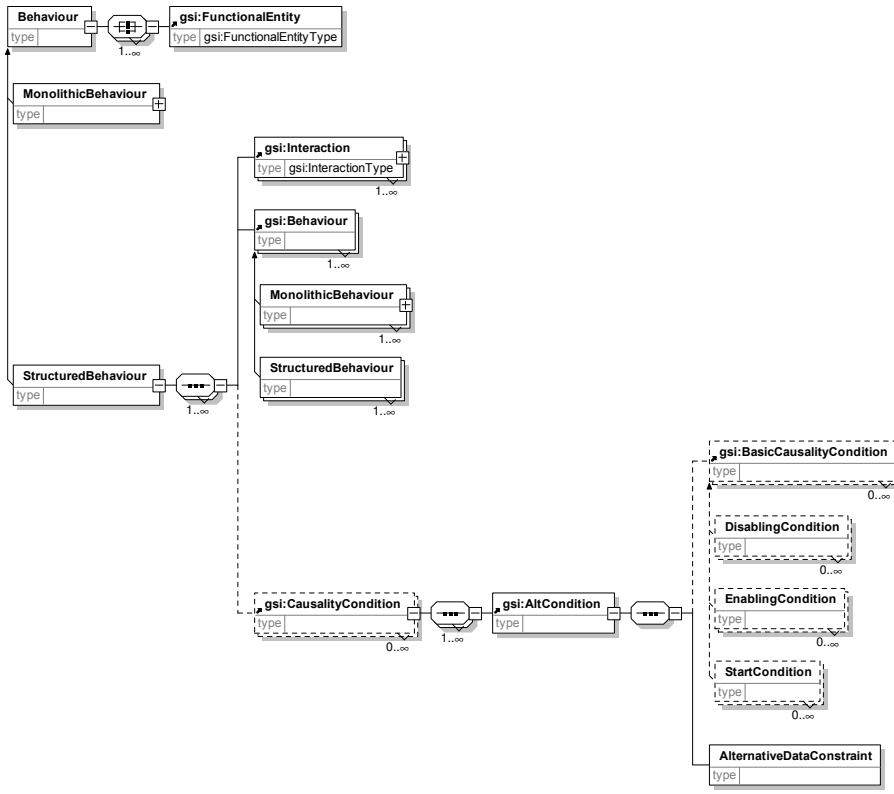


Figure B.3: Repository schema - part III

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ============================== -->
<!--     GSI Repository Schema      -->
<!-- ============================== -->
<!--    International Institute     -->
<!--  for Geo-Information Science   -->
<!--             and                -->
<!--    Earth Observation (ITC)     -->
<!-- ============================== -->
<!--   University of Twente (UT)    -->
<!--      Centre for Telematics     -->
<!--  and Information Technology    -->
<!-- ============================== -->
<!--   Enschede, The Netherlands    -->
<!-- ============================== -->
<!-- ============jmmg============== -->
<!-- ============================== -->

<xs:schema
  targetNamespace="http://www.itc.nl/morales/gsi/schemas"
  xmlns:gsi="http://www.itc.nl/morales/gsi/schemas"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
xmlns:gml="http://www.opengis.net/gml"
elementFormDefault="qualified"
attributeFormDefault="unqualified">

<xs:import
  namespace="http://www.opengis.net/gml"
  schemaLocation="http://www.opengis.net/gml/feature.xsd"/>

<!-- ============= -->
<!-- Main Elements -->
<!-- ============= -->

<xs:element name="Service">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="gsi:Behaviour" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:group name="ElementGroup">
  <xs:sequence>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="gsi:Behaviour"/>
    </xs:choice>
  </xs:sequence>
</xs:group>

<xs:element name="Behaviour">
  <xs:complexType>
    <xs:all maxOccurs="unbounded">
      <xs:element ref="gsi:FunctionalEntity"/>
    </xs:all>
    <xs:attribute name="eID" type="xs:ID"/>
    <xs:attribute name="eName" type="xs:string"/>
  </xs:complexType>
</xs:element>

<xs:element
  name="MonolithicBehaviour"
  type="gsi:MonolithicBehaviourType"
  substitutionGroup="gsi:Behaviour"/>

<xs:element
  name="StructuredBehaviour"
  substitutionGroup="gsi:Behaviour">
  <xs:complexType>
    <xs:sequence maxOccurs="unbounded">
      <xs:element ref="gsi:Behaviour" maxOccurs="unbounded"/>
      <xs:element ref="gsi:Interaction" maxOccurs="unbounded"/>
      <xs:element ref="gsi:CausalityCondition"
                  minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="ActivityUnit">
  <xs:complexType>
```

```
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="gsi:Attribute"/>
    </xs:sequence>
    <xs:attribute name="eID" type="xs:ID"/>
    <xs:attribute name="eName" type="xs:string"/>
  </xs:complexType>
</xs:element>

<xs:element name="Action" substitutionGroup="gsi:ActivityUnit"/>
<xs:element name="Interaction" type="gsi:InteractionType"/>
<xs:element name="InteractionContribution" substitutionGroup="gsi:ActivityUnit"/>

<xs:element name="CausalityCondition">
  <xs:complexType>
    <xs:sequence maxOccurs="unbounded">
      <xs:element ref="gsi:AlternativeCausalityCondition"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="BasicCausalityCondition"/>

<xs:element name="StartCondition"
            substitutionGroup="gsi:BasicCausalityCondition"/>

<xs:element name="EnablingCondition"
            substitutionGroup="gsi:BasicCausalityCondition"/>

<xs:element name="DisablingCondition"
            substitutionGroup="gsi:BasicCausalityCondition"/>

<xs:element name="Attribute">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="gsi:DataType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="InformationAttribute" substitutionGroup="gsi:Attribute"/>
<xs:element name="LocationAttribute" substitutionGroup="gsi:Attribute"/>
<xs:element name="TimeAttribute" substitutionGroup="gsi:Attribute"/>
<xs:element name="Item" substitutionGroup="gsi:InformationAttribute"/>

<xs:element name="DataType">
  <xs:complexType>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="gsi:DataTypeAttribute"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="DataTypeAttribute">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Value" type="gsi:anyDataType"/>
    </xs:sequence>
```

```
  </xs:complexType>
</xs:element>

<xs:element name="PrimitiveDataType" type="xs:anyType"
            substitutionGroup="gsi:DataType"/>

<xs:element name="DefinedDataType" substitutionGroup="gsi:DataType"/>
<xs:element name="FunctionalEntity" type="gsi:FunctionalEntityType"/>
<xs:element name="geoFeature" substitutionGroup="gsi:DefinedDataType"/>

<xs:element name="geoObject" substitutionGroup="gsi:geoFeature">
  <xs:complexType>
    <xs:choice maxOccurs="unbounded">
      <xs:sequence maxOccurs="unbounded">
        <xs:element ref="gml:_Geometry"/>
      </xs:sequence>
      <xs:sequence>
        <xs:element name="Raster" type="gsi:Image"/>
      </xs:sequence>
      <xs:element name="Metadata" type="gsi:MetadataType"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name="Theme" substitutionGroup="gsi:geoFeature">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="gsi:geoObject" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="Semantic" type="xs:string"/>
  </xs:complexType>
</xs:element>

<xs:element name="Collection">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="gsi:Theme" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Composite" substitutionGroup="gsi:geoFeature">
  <xs:complexType>
    <xs:all maxOccurs="unbounded">
      <xs:element ref="gsi:geoObject"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<!-- =============== -->
<!-- Primitive Types -->
<!-- =============== -->

<xs:complexType name="ElementContainer">
  <xs:choice>
    <xs:group ref="gsi:ElementGroup"/>
  </xs:choice>
</xs:complexType>
```

```
<xs:complexType name="MonolithicBehaviourType">
  <xs:complexContent>
    <xs:extension base="gsi:AbstractElementType">
      <xs:sequence>
        <xs:element ref="gsi:CausalityCondition" minOccurs="0"/>
        <xs:element ref="gsi:ActivityUnit" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="InteractionType">
  <xs:complexContent>
    <xs:extension base="gsi:AbstractElementType">
      <xs:sequence>
        <xs:element ref="gsi:InteractionContribution"
                    minOccurs="2" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="anyDataType"/>
<xs:complexType name="Image"/>

<xs:complexType name="FunctionalEntityType">
  <xs:complexContent>
    <xs:extension base="gsi:AbstractElementType"/>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="MetadataType">
  <xs:sequence>
    <xs:element name="Identifier" maxOccurs="unbounded"/>
    <xs:element name="Creator"/>
    <xs:sequence>
      <xs:element name="Content" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="CollectionDate" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="SpatialExtent" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="Quality" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="ReferenceSystem" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="SpatialResolution" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="TemporalResolution" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="SpatialRepresentation" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="Distribution" minOccurs="0"/>
    </xs:sequence>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="BehaviourType">
  <xs:complexContent>
    <xs:extension base="gsi:AbstractElementType"/>
  </xs:complexContent>
</xs:complexType>

<xs:element name="AlternativeCausalityCondition">
  <xs:complexType>
```

```
      <xs:sequence>
        <xs:element ref="gsi:BasicCausalityCondition"
                    minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="AlternativeDataConstraint">
          <xs:complexType>
            <xs:attribute name="expression" type="xs:string"/>
            <xs:attribute name="result" type="xs:anySimpleType"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- ============== -->
  <!-- Abstract Types -->
  <!-- ============== -->

  <xs:complexType name="AbstractElementType">
    <xs:attribute name="eID" type="xs:ID"/>
    <xs:attribute name="eName" type="xs:string"/>
  </xs:complexType>
</xs:schema>
```

Appendix *C*

GML Overview

We adopted GML for the encoding of geographic features, therefore, here we provide a short description of the most important features of GML. The Geography Markup Language (GML) is an XML encoding for the transport and storage of geographic information, including both the spatial and non-spatial properties of geographic features. The GML specification [OGC03a] defines the XML Schema syntax, mechanisms, and conventions that:

- Enable the creation and maintenance of linked geographic application schemas and datasets;
- Support the storage and transport of application schemas and data sets;
- Support the description of geospatial application schemas for specialised domains and information communities;
- Increase the ability of organisations to share geographic application schemas and the information they describe.
- Provide an open, vendor-neutral framework for the definition of geospatial application schemas and objects;
- Allow profiles that support proper subsets of GML framework descriptive capabilities;

GML was developed with a number of explicit design goals, a few of which overlap the objectives of XML itself:

- provide a means of encoding spatial information for both data transport and data storage, especially in a wide-area Internet context;
- be sufficiently extensible to support a wide variety of spatial tasks, from portrayal to analysis;

- establish the foundation for Internet GIS in an incremental and modular fashion;
- allow for the efficient encoding of geo-spatial geometry (e.g. data compression);
- provide easy-to-understand encodings of spatial information and spatial relationships;
- be able to separate spatial and non-spatial content from data presentation (graphic or otherwise);
- permit the easy integration of spatial and non-spatial data, especially for cases in which the non-spatial data is XML-encoded;
- be able to readily link spatial (geometric) elements to other spatial or non-spatial elements.
- provide a set common geographic modeling objects to enable interoperability of independently-developed applications.

The encoding of spatial features using GML requires is based on two XML Schemas: the GML Feature Schema and the GML Geometry Schema; with these two simple schemas it is possible to encode a wide variety of geospatial information.

The GML Geometry schema includes type definitions for both abstract geometry elements, concrete (multi) point, line and polygon geometry elements, as well as complex type definitions for the underlying geometry types. Figures C.1 and C.3 depict the GML Geometry schema.

In the feature schema, the link of a feature with a geometry is modeled as an association class called, geometric property. The Feature schema uses the ¡include¿ element to bring in the definitions and declarations contained in the Geometry schema, and use them for the definitions of the features' geometry property. Figure C.2 shows a representation of the GML Feature schema.

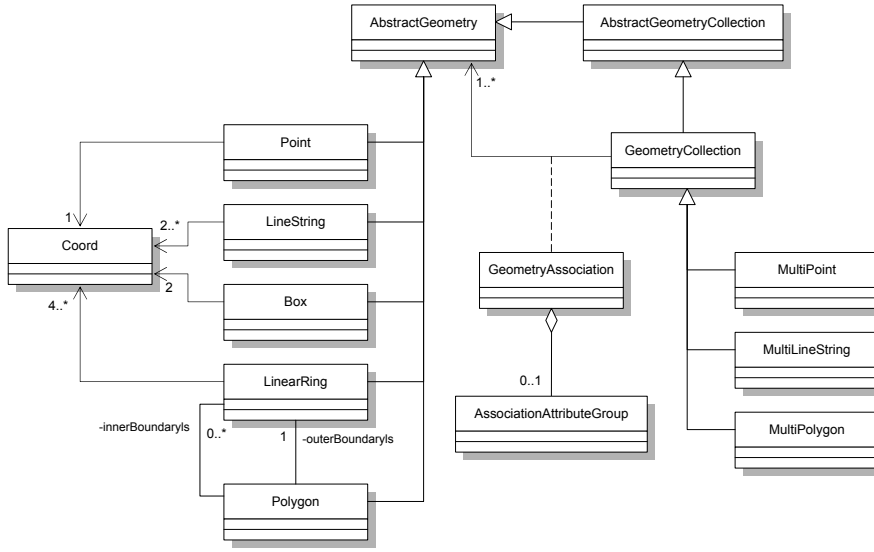


Figure C.1: GML Geometry schema (class representation)

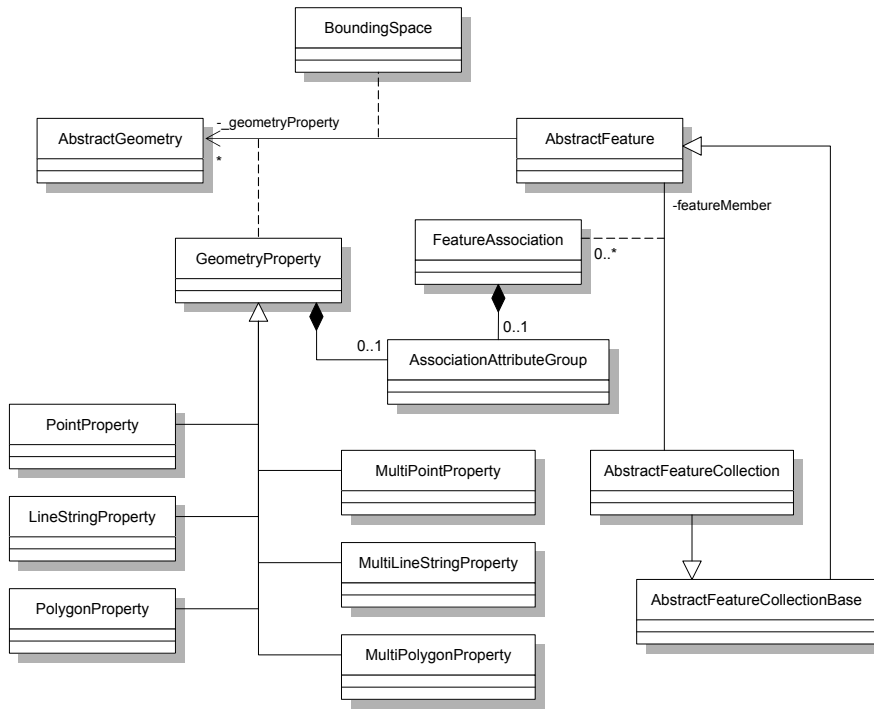


Figure C.2: GML Feature schema (class representation)

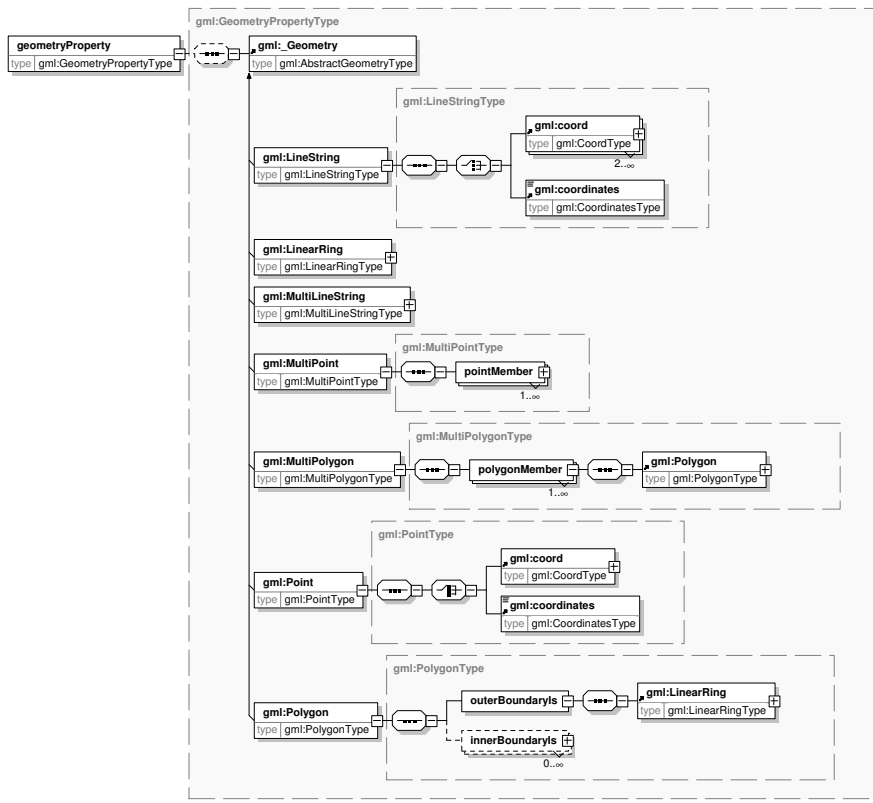


Figure C.3: GML Geometry schema

Bibliography

[AC03] Luisa Liliana Alvarez Casallas. *Geoinformation Virtual Enterprises Design and Process Management.* Master's thesis, International Institute for Geo-Information Science and Earth Observation ITC, Enschede, The Netherlands, March 2003.

[ACD+03] Tony Andrews, Francisco Curbera, Hitesh Dholakia et al. *Business Process Execution Language: for Web Services (BPEL4WS).* BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems., May 2003. Version 1.1.

[AD97] T. Adams and S. Dworkin. *WfMC Workflow Handbook*, chapter Workflow Interoperability Between Businesses, pages 211–221. John Wiley & Sons, 1997.

[Ade01] Martin Ader. WfMC*: Workflow handbook 2001*, chapter Technologies for the Virtual Enterprise, pages 19–38. Future Strategies Inc., Florida, United States of America, 2001.

[AG96] Robert Allen and David Garlan. 'A case study in architectural modelling: The aegis system.' In *Proceedings of the Eighth International Workshop on Software Specification and Design (IWSSD-8)*, pages 6–15. Paderborn, Germany, March 1996.

[AHPW97] G. Abeysinghe, P. Henderson, K. Phalp and R. Walters. 'An audience centred approach to modelling for business process reengineering.' In *5th International Conference on ReTechnologies for Information Systems (ReTIS 97).* Klagenfurt, Austria, 1997.

[AIG03] 'Agent-based Imagery and Geospatial Processing Architecture (AIGA).' aiga.cs.gmu.edu, accessed on January 2003.

[Ala01] N. Alameh. *Scalable and Extensible Infrastructures for Distributing Interoperable Geographic Information Services on the Internet.* Ph.D. thesis, Massachusetts Institute of Technology (MIT), Massachussets, United States of America, 2001.

[All97] Robert J. Allen. *A Formal Approach to Software Architecture*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pitsburg, United States of America, May 1997. CMU-CS-97-144.

[AN01] Jim Arlow and Ila Neustadt. *UML and the Unified Process: Practical Object-Oriented Analysis and Design*. Object Technology Series. Addison-Wesley Pub. Co., Reading, Massachusetts, 1^{st} edition, 2001. ISBN 0-201-77060-1.

[ANZ02] 'The spatial information council (ANZLIC).' www.anzlic.org.au, accessed on October 2002. Australia & New Zealand.

[Aro95] Stan Aronoff. *Geographic Information Systems: A Management Perspective*. WDL Publications, Ottawa, Canada, 1995. ISBN 0-92184-91-1.

[ARR96] Colin G. Armistead, Philip Rowland and A. P. Rowland. *Managing Business Processes: BPR and Beyond*. John Wiley & Sons Ltd., London, United Kingdom, October 1996. ISBN 0-471-95490-X.

[BB90] Barry Boehm and Frank Belz. 'Experiences with the spiral model as a process model generator.' In *Proceedings of the* 5^{th} *international software process workshop on Experience with software process models*, pages 43–45. IEEE Computer Society Press, Kennebunkport, Maine, United States of America, 1990.

[BB94] O. Biberstein and D. Buchs. 'An object-oriented specification language based on hierarchical algebraic Petri nets.' In *Working papers of the international Workshop on Information System Correctness and Reusability IS-CORE'94*, edited by R. Wieringa and R. Feenstra, pages 47–62. Amsterdam, The Netherlands, 1994.

[BB95] Alex Bakman and Alexander Bakman. *How to Deliver Client/Server Applications That Work*. Prentice-Hall International, Great Britain, 1995. ISBN 0-13-304601-X.

[BBG97] Olivier Biberstein, Didier Buchs and Nicolas Guelfi. 'Object-oriented nets with algebraic specifications: The CO-OPN/2 formalism.' In *Advances in Petri Nets on Object-Orientation*, edited by G. Agha and F. De Cindio. Springer-Verlag, 1997.

[BBS+02] Christoph Brox, Yaser Bishr, Kristian Senkler, Katharina Zens and Werner Kuhn. 'Toward a geospatial data infrastructure for Northrhine-Westphalia.' *Computers, Environment and Urban Systems*, Vol. 26(1):19–37, 2002.

[BCE+02] Tom Bellwood, Luc Clément, David Ehnebuske et al. Universal Description, Discovery and Integration (UDDI). Published specification, Accenture, Ariba, Inc., Commerce One, Inc., Fujitsu Limited, Hewlett-Packard Company, i2 Technologies, Inc., Intel Corporation, International Business Machines Corporation, Microsoft Corporation, Oracle Corporation, SAP AG, Sun Microsystems, Inc., and VeriSign, Inc., July 2002. Version 3.0.

[BCK03] Len Bass, Paul Clements and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Pub. Co., Reading, Massachusetts, 2^{nd} edition, April 2003. ISBN 0-321-15495-9.

[Ber96] Alex Berson. *Client/Server Architecture*. McGraw Hill, New York, United States of America, 2^{nd} edition, 1996. ISBN 0-07-005664-1.

[BFH03] Fran Berman, Geoffrey Fox and Tony Hey. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley Series in Communications & Distributed Systems. John Wiley & Sons Ltd., West Sussex, England, April 2003. ISBN 0-470-85319-0.

[BG00] Didier Buchs and Nicolas Guelfi. 'A formal specification framework for object-oriented distributed systems.' *IEEE Transactions on Software Engineering*, Vol. 26(No. 7):635–651, July 2000.

[Bib97] Olivier Biberstein. *An Object-Oriented Formalism for the Specification of Concurrent Systems*. Ph.D. thesis, University of Geneva, Genéve, Switzerland, July 1997. Ph.D. Thesis, no. 2919.

[Bis97] Yaser Bishr. *Semantic Aspects of Interoperable GIS*. Ph.D. thesis, Wageningen Agricultural University, Wageningen, The Netherlands, November 1997. ITC Dissertation No. 47.

[BKR99] Y. Bishr, W. Kuhn and M. Radwan. *Interoperating Geographic Information Systems*, chapter Probing the semantic content of information communities – a first step toward semantic interoperability, pages 211–221. Kluwer Academic Publishers, 1999. ISBN 0792384369.

[BLHL01] Tim Berners-Lee, James Hendler and Ora Lassila. 'The semantic web.' *Scientific American*, May 2001.

[BN96] Peter Bernus and Laszlo Nemes. 'Modelling and methodologies for enterprise integration.' In *Proceedings of the International Federation for Information Processing (IFIP) Conference on Models and Methodologies for Enterprise Integration*. Chapman & Hall, 1996.

[Boe88] Barry W. Boehm. 'A spiral model of software development and enhancement.' In *Computer*, volume 21, pages 61–72. IEEE Computer Society Press, 1988.

[Boe96] Barry Boehm. 'Anchoring the software process.' *IEEE Software*, Vol. 13(No. 4):73–82, July 1996.

[Boo94] Grady Booch. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley Pub. Co., 2^{nd} edition, February 1994. ISBN 0-8053-5340-2.

[Bra94] Michael H. Brackett. *Data Sharing: Using A Common Data Architecture*. John Wiley & Sons Ltd., New York, United States of America, March 1994. ISBN 0-471-30993-1.

[Bra01] Eric J. Braude. *Software Engineering: An Object-Oriented Perspective*. John Wiley & Sons Ltd., New York, United States of America, 2001. ISBN 0-471-32208-3.

[Bro04] Alan Brown. 'An introduction to Model Driven Architecture part I: MDA and todays systems.' *The Rational Edge*, February 2004.

[BRP97] Yaser Bishr, Mostafa Radwan and J. Pandya. 'SemWeb - a prototype for seamless sharing of geoinformation on the World Wide Web in a client/server architecture.' In *Proceedings of the Joint European Conference and Exhibition on Geographical Information*, volume Vol. 1, pages 145–154. Vienna, Austria, April 1997.

[BSC94] Rosalinda Barden, Susan Stepney and David Cooper. *Z in Practice*. Prentice-Hall International, Great Britain, 1994. ISBN 0-13-124934-7.

[By91] Rolf A. de By. *The Integration of Specification Aspects in Database Design*. Ph.D. thesis, University of Twente, Enschede, The Netherlands, October 1991.

[By01] Rolf A. de By (editor). *Principles of Geographic Information Systems*. ITC Educational Textbook Series. International Institute for Geo-Information Science and Earth Observation ITC, Enschede, The Netherlands, 2001. ISBN 90-6164-184-5.

[Cai96] Luís Caires. 'A language for the logical specification of processes and relations.' In *Proceedings of the Algebraic and Logic Programming International Conference ALP'96*, edited by Michael Hanus, volume 6, pages 150–164. Springer-Verlag, 1996.

[CD00] John Cheesman and John Daniels. *UML Components: A Simple Process for Specifying Component-Based Software*. The Component Software Series. Addison-Wesley Pub. Co., New Jersey, United States of America, 2000. ISBN 0-201-70851-5.

[CDK01] George Coulouris, Jean Dollimore and Tim Kindberg. *Distributed Systems: Concepts and Design*. International Computer Science Series. Addison-Wesley Pub. Co., Harlow, England, 3^{rd} edition, 2001. ISBN 0201-61918-0.

[CFFK01] Karl Czajkowskiy, Steven Fitzgeraldz, Ian Fosterx and Carl Kesselman. 'Grid information services for distributed resource sharing.' In *Proceedings of the 10^{th} IEEE Symposium on High Performance Distributed Computing*, pages 181–194. IEEE Press, August 2001.

[Cod70] E. F. Codd. 'A relational model of data for large shared data banks.' *Communications of the ACM*, Vol. 13(No. 6):377–387, June 1970. Association for Computing Machinery, Inc.

[CZ96] Edward Chan and Rupert Zhu. 'Ql/g - a query language for geometric data bases.' In *Proceedings of the 1st International Conference on GIS in Urban Regional and Environment Planning*, pages 271–286. Samos, Greece, April 1996.

[DAM03] 'Darpa Agent Markup Language for Services (DAML-S).' www.daml.org/services, accessed on September 2003.

[Dav93] Thomas H. Davenport. *Process Innovation: Reengineering Work Through Information Technology.* Harvard Business School Press, Boston, Massachusetts, 1993. ISBN 0-87584-366-2.

[Dav99] Georgia Davanelou. *Simulation of Business Process Scenarios for the Evaluatin of Cadastral Operations.* Master's thesis, International Institute for Geo-Information Science and Earth Observation ITC, Enschede, The Netherlands, February 1999.

[DCM03a] 'Dublin Core Metadata Initiative.' dublincore.org, accessed on September 2003.

[DCM03b] 'The Semantic Web community portal.' www.semanticweb.org, accessed on November 2003.

[DDT02] H. M. Deitel, P. J. Deitel B. Duwaldt and L. K. Trees. *Web Services: A Technical Introduction.* Deitel^{TM} Developer Series. Prentice-Hall International, New Jersey, United States of America, August 2002. ISBN 0-13-046135-0.

[Dem00] Michael N. Demers. *Fundamentals od Geographic Information Systems.* John Wiley & Sons Ltd., New York, United States of America, 2nd edition, 2000. ISBN 0-471-31423-4.

[DH00] Scott A. DeLoach and Thomas C. Hartrum. 'A theory-based representation for object-oriented domain models.' *IEEE Transactions on Software Engineering*, Vol. 26(No. 6):500–517, June 2000.

[DHM+01] G. Denker, J.R. Hobbs, D. Martin, S. Narayana and R. Waldinger. 'Accessing information and services on the daml-enabled web.' In *The Second International Workshop on the Semantic Web.* 2001.

[Dil94] Antoni Diller. *Z An introduction to Formal Methods.* John Wiley & Sons Ltd., Chichester, England, 2nd edition, 1994. ISBN 0-471-93973-0.

[DM99] Giovana Di Marzo. *Stepwise Refinment of Formal Specifications on Logical Formulae: From CO-OPN/2 specifications to Java programs.* Ph.D. thesis, Department d'Informatique, Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland, January 1999. Ph.D. Thesis, no. 1931.

[Dou01] Nebert D. Douglas. *Developing Spatial Data Infrastructures: The SDI Cookbook.* Technical report, Global Spatial Data Infrastructure Organization, May 2001.

[D'S01] Desmond D'Souza. 'Model-driven architecture and integration: Opportunities and challenges.' www.kinetium.com, March 2001.

[Dur92] R. C. J. Dur. *Business Reengineering in Information Intensive Organisations.* Ph.D. thesis, Technical University Delft, Delft, The Netherlands, 1992.

[DW99] Desmond Francis D'Souza and Alan Cameron Wills. *Objects, Components and Frameworks with UML: The Catalysis Approach.* The Addison-Wesley object technology series. Addison-Wesley Pub. Co., Reading, Massachusetts, 1999. ISBN 0-201-31012-0.

[EJL+99] Henk Eertink, Wil Janssen, Paul Oude Luttighuis, Wouter B. Teeuw and Chris A. Vissers. 'A business process design language.' In *World Congress on Formal Methods (1)*, pages 76–95. 1999.

[EN00] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems.* Addison-Wesley Pub. Co., Reading, Massachusetts, 3^{rd} edition, 2000. ISBN 0-8053-1755-4.

[EP98] Hans-Erik Eriksson and Magnus Penker. *UML Toolkit.* John Wiley & Sons Ltd., Toronto, Canada, 1998. ISBN 0-471-19161-2.

[Far02] Cléver Ricardo Guareis de Farias. *Architectural Design of Groupware Systems: a Component-Based Approach.* Ph.D. thesis, University of Twente, Enschede, The Netherlands, May 2002. Ph.D. Thesis, no. 01-38.

[FGD98] *Content Standard for Digital Geospatial Metadata (CSDGM).* FGDC-STD-001-1998, Federal Geographic Data Committee (FGDC), Washington, D.C., June 1998.

[FGD03] 'National Spatial Data Infrastructure (NSDI).' www.fgdc.gov, accessed on February 2003. Federal Geographic Data Committee (FGDC), United States of America.

[FH93] Peter Feiler and Watts Humphrey. 'Software process development and enactment concepts and definitions.' In *Proceedings of the* 2^{nd} *International Conference on the Software Process*, pages 28–40. IEEE Computer Society Press, Berlin, Germany, February 1993.

[Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* Ph.D. thesis, University of California, Irvine, California, 2000.

[Fis00] Layna Fischer. *Excellence in Practice: Innovation and Excellence in Workflow Process and Knowledge Management*, volume III. Future Strategies Inc., Lighthouse Point, Florida, 2000. ISBN 0-9640233-8-5.

[Fis01] —. WfMC*: Workflow handbook 2001.* Future Strategies Inc., Florida, United States of America, 2001. ISBN 0-9703509-0-2.

[FP94] Luís Ferreira Pires. *Architectural Notes: a Framework for Distributed Systems Development*. Ph.D. thesis, University of Twente, Enschede, The Netherlands, September 1994. Ph.D. Thesis, no. 94-01.

[FPSFAA03] Luís Ferreira Pires, Marten van Sinderen, Cléver Ricardo Guareis de Farias and João Paulo Andrade Almeida. *Use of Models and Modelling Techniques for Service Development*. Centre for Telematics and Information Technology, University of Twente, Enschede, The Netherlands, September 2003. Technical report.

[Fra96] L. Franken. *Quality of Service Management: A Model-Bases Approach*. Ph.D. thesis, Centre for Telematics and Information Technology, Twente University, Enschede, The Netherlands, 1996.

[FS00] Martin Fowler and Kendall Scott. *UML Distilled: A Brief Guide to the Standard Object Modelling Language*. Object Technology Series. AWpc, New Jersey, United States of America, $2^{nd}0$ edition, 2000. ISBN 0-201-65783-X.

[FSJ99] Mohamed E. Fayad, Douglas C. Smith and Ralph E. Johnson. *Implementing Application Frameworks: Object-Oriented Frameworks at Work*. John Wiley & Sons Ltd., New York, United States of America, 1999. ISBN 0-471-25201-8.

[FWQFP97] H. M. Franken, M. K. de Weger, D. A. C. Quartel and L. Ferreira Píres. 'On engineering support for business process modelling and redesign.' In *Modelling Techniques for Business Process Re-engineering and Benchmarking*, edited by Guy Doumeingts and Jim Browne, chapter 10, pages 103–120. Chapman & Hall, London, UK, 1^{st} edition, 1997. ISBN 0-412-78910-8.

[Gar95] David Garlan. 'What is style?' In *Proceedings of the Dagstuhl Workshop on Software Architecture*. Saarbruecken, Germany, February 1995.

[GDI02] 'Geodateninfrastruktur Nordrhein-Westfalen (GDI-NRW).' gdi-nrw.uni-muenster.de/index.html, accessed on December 2002.

[GHS95] Dimitrios Georgakopoulos, Mark F. Hornick and Amit P. Sheth. 'An overview of workflow management: From process modelling to workflow automation infrastructure.' In *Distributed and Parallel Databases*, volume 3, pages 119–153. Kluwer Publishers, United States of America, 1995.

[GM00] Richard Groot and John McLaughlin. *Geospatial Data Infrastructure: Concepts, cases and good practice*. Oxford University Press, New York, United States of America, 2000. ISBN 0-19-823381-7.

[Gra01] Ian Graham. *Object-Oriented Methods: Principles and Practice*. The Addison-Wesley object technology series. Addison-Wesley Pub. Co., Reading, Massachusetts, 3^{rd} edition, 2001. ISBN 0-201-61913-X.

[GS93] David Garlan and Mary Shaw. 'An introduction to software architecture.' In *Advances in Software Engineering and Knowledge Engineering*, edited by V. Ambriola and G. Tortora, pages 1–39. World Scientific Publishing Company, Singapore, 1993.

[GTM99] James E. Goldman, Rawles Phillip T. and Julie R. Mariga. *Client/Server Information Systems: A Business-Oriented Apporach*. John Wiley & Sons Ltd., New York, United States of America, 1999. ISBN 0-471-29654-6.

[Haw01] Igor Hawryszkiewycz. *Introduction to Systems Analysis and Design*. Prentice-Hall International, 5^{th} edition, February 2001. ISBN 1740092805.

[HBCS03] Richard Hull, Michael Benedikt, Vassilis Christophides and Jianwen Su. 'E-services: A look behind the curtain.' In *Proceedings of the twenty second ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems*, pages 1–14. ACM Press, San Diego, California, June 2003.

[HC01] Michael Hammer and James A. Champy. *Reengineering the Corporation: A Manifesto for Business Revolution*. HarperBusiness, New York, United States of America, revised edition, June 2001. ISBN 0-06-662112-7.

[Hee94] K. M. van Hee. *Information System Engineering: a Formal Approach*. Cambridge University Press, New York, United States of America, 1994. ISBN 0-521-45514-6.

[HENN97] H. James Harrington, Erik K. C. Esseling, Harm van Nimwegen and Nico van Nimwegen. *Business Process Improvement Workbook: Documentation, Analysis, Design, and Management of Business Process Improvement*. McGraw-Hill Trade, April 1997. ISBN 0-07-026779-0.

[HGS95] Ralf Hartmut Güting and Markus Schneider. 'Realm-based spatial data types: The ROSE algebra.' *VLDB Journal*, Vol. 4(No. 2):243–286, 1995.

[Hil99] Rich Hilliard. 'Views and viewpoints in software systems architecture.' In *First Working IFIP Conference on Software Architecture*. San Antonio, 1999.

[HK01] H. Hayami and M. Katsumata. WfMC*: Workflow handbook 2001*, chapter Interworkflow: A challenge for Business-to-Business Electronic Commerce. Future Strategies Inc., Florida, United States of America, 2001. ISBN 0-9703509-0-2.

[Hoq00] Faisal Hoque. *e-Enterprise: Business, Models,Architectues and Components*. Breakthroughs in Application Development Series. Cambridge University Press, New York, United States of America, 2000. ISBN 0-521-77487-X.

[ICD02] 'Infraestructura Colombiana de Datos Espaciales (ICDE).' codazzi4.igac.gov.co/icde, accessed on June 2002. Colombia.

[IEE00] IEEE Computer Society. *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*. Technical report, The Institute of Electrical and Electronics Engineers, Inc., September 2000.

[ISO96a] ISO/IEC. *10746-2: Information Technology – Open Distributed Processing – Reference Model: Foundations*. Draft International Standard, International Organization for Standardization, Genève, Switzerland, September 1996.

[ISO96b] —. *10746-3: Information Technology – Open Distributed Processing – Reference Model: Architecture*. Draft International Standard, International Organization for Standardization, Genève, Switzerland, December 1996.

[ISO98a] —. *10746-1: Information Technology – Open Distributed Processing – Reference Model: Overview*. Draft International Standard, International Organization for Standardization, Genève, Switzerland, December 1998.

[ISO98b] —. *10746-4: Information Technology – Open Distributed Processing – Reference Model: Architecture semantics*. Draft International Standard, International Organization for Standardization, Genève, Switzerland, December 1998.

[ISO01a] *ISO/DIS 19110: Geographic information – Feature cataloguing methodology*. Draft International Standard, ISO/TC 211, Genève, Switzerland, December 2001.

[ISO01b] *ISO/DIS 19119: Geographic information – Services*. Draft International Standard, ISO/TC 211, Genève, Switzerland, December 2001.

[ISO01c] *ISO/FDIS 19115: Geographic information – Metadata*. Final Draft International Standard, ISO/TC 211, Genève, Switzerland, December 2001.

[ISO02] *ISO 19101: Geographic information – Reference model*. International Standard, ISO/TC 211, Genève, Switzerland, December 2002.

[ISO03] 'ISO 639.2 – codes for the representation of names of languages.' www.loc.gov/standards/iso639-2/langhome.html, accessed on November 2003.

[JBR99] Ivar Jacobson, Grady Booch and James Rumbaugh. *The Unified Software Development Process*. The Addison-Wesley object technology series. Addison-Wesley Pub. Co., Reading, Massachusetts, 1999. ISBN 0-201-57169-2.

[JCJ92] Ivar Jacobson, Magnus Christerson and Patrik Jonsson. *Object-Oriented Software Engineering - A Use Case Driven Approach*. Addison-Wesley Pub. Co., Boston, Massachusetts, June 1992. ISBN 0-201-54435-0.

[JF96] Henk Jonkers and Henry M. Franken. 'Quantitative modelling and analysis of business processes.' In *Simulation in Industry: Proceedings of the* 8^{th} *European Simulation Symposium*, volume Vol. I, pages 175–179. Bruzzone and E.Ker-ckho's eds., 1996.

[JOS94] Peter Jaescje, Andreas Oberweis and Wolffried Stucky. 'Deriving complex structured object types for business process modelling.' In *Proceedings of the 13th International Conference on the Entity-Relationship Approach*, edited by P. Loucopoulos, pages 28–45. Springer-Verlag, Manchester, United Kingdom, 1994.

[Kar99] James Samuel Karioki. *A Structured Methodology and Implementation Strategy for Business Process Reengineering in Geo-information Production.* Master's thesis, International Institute for Geo-Information Science and Earth Observation ITC, Enschede, The Netherlands, June 1999.

[Kaw91] Peter Kawalek. *Process Modelling Cookbook: Version 1.* Technical report, University of Manchester and British Telecommunications, United Kingdom, September 1991.

[KFM99] J. Kanet, W. Faisst and P. Mertens. 'Application of information technology to a virtual enterprise broker: The case of bill epstein.' *International Journal of Production Economics*, 62:23–32, 1999.

[KG96] David Kinny and Michael Georgeff. 'Modelling and design of multi-agent systems.' In *Intelligent Agents III: Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96). LNAI 1193.* Springer-Verlag: Heidelberg, Germany, Budapest, 1996.

[KH96] Haim Kilov and William Harvey. *Object-Oriented Behavioral Specifications.* Kluwer Academic Publishers, Boston, United States of America, 1996. ISBN 0-7923-9778-9.

[KK97] Peter Kueng and Peter Kawalek. 'Goal-based business process models: creation and evaluation.' *Business Process Management Journal*, Vol. 3(No. 1):17–38, 1997.

[Kot99] C. Kottman. *Interoperating geographic Information Systems*, chapter The OpenGIS Consurtium and progress Toward Interoperability in GIS. Kluwer Academic Publishers, 1999.

[KR03] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Apprpach Featuring the Internet.* Addison-Wesley Pub. Co., New York, United States of America, 2nd edition, 2003. ISBN 0-201-97699-4.

[Kru99] Philippe Kruchten. *The rational unified process.* The Addison-Wesley object technology series. Addison-Wesley Pub. Co., Reading, Massachusetts, 1999. ISBN 0-201-60459-0.

[KVZ99] K. Kosanke, F. Vernadat and M. Zelm. 'CIMOSA: Enterprise engineering and integration.' *Computers in Industry*, Vol. 40(No. 2-3):83–97, 1999.

[KWB03] Anneke Kleppe, Jos Warmer and Wim Bast. *MDA Explained: The Model Driven Architecture Practice and Promise.* The Addison-Wesley object technology series. Addison-Wesley Pub. Co., Reading, Massachusetts, April 2003. ISBN 0-321-19442-X.

[Laa97] P. Laarakker. *IT-2000: Steps to the Future of the Netherlands Cadastre*. Technical report, Cadastre and Public Register Agency, Apeldoorn, The Netherlands, 1997.

[Lar02] Graig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Aanalysis and Design and the Unified Process*. Prentice-Hall International, New Jersey, United States of America, 2^{nd} edition, 2002. ISBN 0-13-092569-1.

[Lew94] Paul Lewis. *Information-Systems Development*. Pitman Publishing, London, England, 1994. ISBN 0-273-03107-4.

[LGMR01] Paul A. Longley, Michael F. Goodchild, David J. Maguire and David W. Rhind. *Geographic Information Systems and Science*. John Wiley & Sons Ltd., West Sussex, England, 2001. ISBN 0-471-89275-0.

[Lig01] David Lightfoot. *Formal Specification Using Z*. Grassroots. Palgrave Publishers Ltd., New York, United States of America, 2^{nd} edition, 2001. ISBN 0-333-76327-0.

[LK99] Thomas M. Lillesand and Ralph W. Kiefer. *Remote Sensing and Image Interpretation*. John Wiley & Sons Ltd., New York, United States of America, 4^{th} edition, 1999. ISBN 0-471-25515-7.

[LL02] Kenneth C. Laudon and Jane P. Laudon. *Essentials of Management Information Systems*. Prentice-Hall International, 5^{th} edition, May 2002. ISBN 0-13-008734-3.

[Mar00] Chris Marshall. *Enterprise Modelling with UML*. The Addison-Wesley object technology series. Addison-Wesley Pub. Co., Boston, Massachusetts, 2000. ISBN 0-201-43313-3.

[MB02] Stephen J. Mellor and Marc J. Balcer. *Executable UML: A Foundation for Model Driven Architecture*. Object Technology Series. Addison-Wesley Pub. Co., Reading, Massachusetts, May 2002.

[MFPS02] Javier Morales, Luís Ferreira Pires and Marten van Sinderen. 'Model driven geo-information systems development.' In *Proceedings of the Sixth International Enterprise Distributed Object Computing Conference EDOC 2002*, pages 155–166. IEEE Computer Society, Lausanne, Switzerland, September 2002.

[MG01] Javier M. Morales G. 'On the design of geoinformation provision systems.' In *Proceedings of the 5^{th} Global Spatial Data Infrastructure Conference*. Cartagena, Colombia, May 2001.

[MJ82] D. D. McCraken and M. A. Jackson. 'Life cycle concept considered harmful.' *ACM Software Engineering Notes*, Vol. 7(No. 2):28–32, 1982.

[MMP+95] Richard J. Mayer, Christopher P. Menzel, Michael K. Painter et al. *Information Integration for Concurrent Engineering (IICE): IDEF3 Process Description Capture Method Report.* Technical report, Knowledge Based Systems, Incorporated, September 1995.

[MOL+80] Harlan D. Mills, D. O'Neill, R. C. Lynger, M. Dyer and R. E. Quinnan. 'The management of software engineering: Parts i to v.' *IBM Systems Journal*, Vol. 24(No. 2):414–477, 1980.

[Mol98] Martien Molenaar. *An Introduction to the Theory of Spatial Object Modelling for GIS.* Research Monographs in GIS series. Taylor & Francis, West Sussex, England, 1998. ISBN 0-7484-0774-X.

[Mor98] Javier Morales. *A Workflow Oriented Design of "on-line" Geoinformation Services.* Master's thesis, International Institute for Geo-Information Science and Earth Observation ITC, Enschede, The Netherlands, July 1998.

[MR02a] Mark W. Maier and Eberhardt Rechtin. *The Art of Systems Architecting.* CRC Press LLC, Florida, United States of America, 2^{nd} edition, 2002. ISBN 0-8493-0440-7.

[MR02b] Javier Morales and Mostafa Radwan. 'Extending geoinformation services: A virtual architecture for spatial data infrastructures.' In *Proceedings of the Joint International Symposium on GeoSpatial Theory, Processing and Applications of the ISPRS Commission IV.* Ottawa, Canada, July 2002.

[MRS00] Javier Morales, Mostafa Radwan and Rosilah Sani. 'A methodology for architectural modelling of spatial information production processes in the context of business process reengineering.' In *Proceedings of the XIXth congress of the International Society for Photogrammetry and Remote Sensing.* Amsterdam, the Netherlands, July 2000.

[MW85] Merriam-Webster. *Webster's Ninth New Collegiate Dictionary.* Merriam-Webster Inc., Massachusetts, United States of America, 1985. ISBN 0-87779-508-8.

[NGD02] 'National Geospatial Data Framework (NGDF).' www.ngdf.org.uk, accessed on November 2002. United Kingdom.

[Nis99] Nimal Nissanke. *Introductory Logic and Sets foe Computer Scientists.* Addison-Wesley Pub. Co., Essex, England, 1999. ISBN 0-201-17957-1.

[Nol03] James J. Nolan. *An Agent-Based Architecture for Distributed Imagery and Geospatial Computing.* Ph.D. thesis, George Mason University, Fairfax, Virginia, United States of America, April 2003.

[Oak97] Les A. Oakshott. *Business Modelling and Simulation.* Pitman Publishing, August 1997. ISBN 0273612514.

[OGC99] *The OpenGIS Abstract Specification Overview*. Version 4, Open GIS Consortium, Inc., June 1999.

[OGC02] *The OpenGIS Abstract Specification Topic 12: OpenGIS Service Architecture*. Version 4.3, Open GIS Consortium, Inc., January 2002. Editor: George Percivall.

[OGC03a] *Geography Markup Language (GML) 3.0*. OpenGIS implementation specification, OpenGIS Consortium, Inc., January 2003. Editors: Simon Cox, Paul Daisey, Ron Lake, Clemens Portele and Arliss Whiteside.

[OGC03b] *OpenGIS Reference Model*. Version: 0.1.2, No. OGC 03-040, Open GIS Consortium Inc., March 2003. Editor: Kurt Buehler.

[OGC03c] *OpenGIS Web Services Architecture*. Discussion paper, Version: 0.3, No. OGC 03-025, Open GIS Consortium Inc., January 2003. Editor: Joshua Lieberman.

[OMG01a] OMG - Architecture Board. *Model Driven Architecture: A technical Perspective*. Draft version 00-17, Object Management Group, January 2001.

[OMG01b] —. *Model Driven Architecture (MDA)*. Technical report, Object Management Group, July 2001.

[OMG01c] OMG - Object Management Group. *Common Warehouse Metamodel (CWM) Specification*. Technical Report Version 1.0, Object Management Group, February 2001.

[OMG01d] —. *Unified Modeling Language Specification*. Technical Report Version 1.4, Object Management Group, September 2001.

[OMG02] —. *Meta Object Facility (MOF) Specification*. Technical Report Version 1.4, Object Management Group, April 2002.

[OMG03a] OMG - Architecture Board. *MDA Guide*. Version 1.0.1, Object Management Group, January 2003.

[OMG03b] OMG - Object Management Group. *Unified Modeling Language Specification*. Technical Report Version 2.0, Object Management Group, August 2003.

[OMG03c] —. *XML Metadata Interchange (XMI) Specification*. Technical Report Version 2.0, Object Management Group, May 2003.

[Ons02] Harlan J. Onsrud. 'Survey of national and regional spatial data infrastructure activities around the globe.' www.spatial.maine.edu/~onsrud/GSDI.htm, accessed on January 2002.

[Ope04] 'Open GIS consortium.' www.opengis.org, accessed on February 2004.

[OS96] Andreas Oberweis and Peter Sander. 'Information system behavior specification by high-level Petri nets.' *ACM Transactions on Information Systems*, Vol. 14(No. 4):380–420, October 1996.

[ÖV99] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Prentice-Hall International, New Jersey, United States of America, 2^{nd} edition, 1999. ISBN 0-13-659707-6.

[Per97] Dewayne E. Perry. 'Directions in process technology – an architectural perspective.' In *Workshop on Research Directions in Process Technology*. Nancy, France, July 1997.

[Pol00] Rudolf Wolfgang van der Pol. *Knowledge-based Query Formulation in Information Retrieval*. Ph.D. thesis, Maastricht University, Maastricht, the Netherlands, September 2000. Dissertation Series No. 2000-5.

[Pre97] Roger S. Pressman. *Software Engineering: A Practitioners Approach*. McGraw-Hill Series on Computer Science. McGraw Hill, New York, United States of America, 4^{th} edition, 1997. ISBN 0-07-052182-4.

[Put01] Janis R. Putman. *Architechting with RM-ODP*. Software Architecture Series. Prentice Hall PTR, New Jersey, United States of America, 2001. ISBN 0-13-019116-7.

[PW92] Dewayne E. Perry and Alexander L. Wolf. 'Foundations for the study of software architecture.' *ACM SIGSOFT Software Engineering Notes*, Vol. 17(No. 4):40–52, 1992.

[QFPS02] Dick Quartel, Luís Ferreira Pires and Marten van Sinderen. 'On architectural support for behaviour refinement in distributed systems design.' *Transactions of the SDPS Journal of Integrated Design and Process Science*, Vol. 6(No. 1):1–30, March 2002.

[Qua98] Dick Quartel. *Action Relations: Basic design concepts for behaviour modelling and refinement*. Ph.D. thesis, University of Twente, Enschede, The Netherlands, February 1998. Ph.D. Thesis, no. 98-18.

[Qua03] —. 'ISDL meta-model and repository.' Draft, Enschede, The Netherlands, July 2003. Arco project No. Arco/WP1/N001/V01.

[RAOM03] M. Mostafa Radwan, Liliana Alvarez, Richard Onchaga and Javier Morales. 'Designing an integrated enterprise model to support partnerships in the geo-information industry.' In *Proceedings of the* 2^{nd} *Annual Asian Conference and Exhibition in the field of GIS, GPS, Aerial Photography and Remote Sensing (Map Asia 2003)*. Kual Lumpur, Malasya, October 2003.

[RAV02] 'Netwerk voor Geo-informatie (RAVI).' www.ravi.nl/index.htm, accessed on December 2002. The Netherlands.

[RBP+91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy and William Lorenson. *Object-Oriented Modelling and Design*. Englewood Cliffs, Prentice-Hall International, New Jersey, United States of America, 1^{st} edition, 1991. ISBN 0-13-629841-9.

[Rey99] Carla Reyneri. 'Operational building blocks for business process modelling.' *Computers in Industry*, Vol. 40(No. 2-3):115–123, 1999.

[RFC03] 'Tags for the identification of languages, internet rfc 3066.' www.ietf.org/rfc/rfc3066.txt, accessed on November 2003.

[Rig93] Darrell K. Rigby. 'The secret history of process reengineering.' Planning Review, March/April 1993.

[RJB98] James Rumbaugh, Ivar Jacobson and Grady Booch. *The Unified Modelling Language Reference Manual*. Object Technology Series. Addison--Wesley Pub. Co., Reading, Massachusetts, December 1998. ISBN 0-201-30998-X.

[ROM01] Mostafa Radwan, Richard Onchaga and Javier Morales. *A Structural Approach to the Management and Optimization of Geoinformation Processes*. Official publication no. 41, European Organization for Experimental Photogrammetric Research (OEEPE), Frankfurt, Germany, 2001.

[Roy70] W. W. Royce. 'Managing the development of large software systems: concepts and techniques.' In *Proceedings of the IEEE Western Electronic Show and Convention (WESCON)*, pages 1–9. IEEE Computer Society Press, Los Angeles, California, United States of America, 1970.

[Roy87] —. 'Managing the development of large software systems: concepts and techniques.' In *Proceedings of the* 9^{th} *international conference on Software Engineering*, pages 328–338. IEEE Computer Society Press, Monterey, California, United States of America, 1987.

[RS97] Luz Angela Rocha Salamanca. *Applying concepts of business process redesign and operations management in a geoinformation production organization: case study "Instituto Geografico Agustin Codazzi", Colombia*. Master's thesis, International Institute for Geo-Information Science and Earth Observation ITC, Enschede, The Netherlands, June 1997.

[RSV01] Philippe Rigaux, Michel Scholl and Agnés Voisard. *Spatial Databases: with Application to GIS*. Morgan Kaufman Publishers, San Francisco, California, 2001. ISBN 1-55860-588-6.

[San98] Rosilah Sani. *Dynamic Modelling in the Reengineering of Geoinformation Production Processes*. Master's thesis, International Institute for Geo-Information Science and Earth Observation ITC, Enschede, The Netherlands, July 1998.

[Sco02] Kendall Scott. *The Unified Process Explained*. Addison-Wesley Pub. Co., Reading, Massachusetts, 2002. ISBN 0-201-74204-7.

[SH96] Pablo A. Straub and Carlos A. Hurtado. 'Understanding behavior of business process models.' In *Coordination Models and Languages*, pages 440–443. 1996.

[SHL95] Pablo A. Straub and Carlos Hurtado L. *Behavioral Consistency in Business Process Models.* Technical Report RT-PUC-DCC-95-4, Catholic University of Chile, Computer Science Department, May 1995.

[Sho03] Yasser Shohoud. *Real World XML Web Services.* The DevelopMentor Series. Addison-Wesley Pub. Co., Boston, United States of America, 2003. ISBN 0-201-77425-9.

[Sim94] Oliver Sims. *Business Objects: Delivering Cooperative Objects for Client-Server.* The IBM McGraw-Hill Series. McGraw Hill, 1994. ISBN 0-07-707957-4.

[Sin95] Marten van Sinderen. *On the Design of Application Protocols.* Ph.D. thesis, University of Twente, Enschede, The Netherlands, March 1995. Ph.D. Thesis, no. 95-04.

[SNI02] 'Centro Nacional de Informação Geográfica (SNIG).' snig.igeo.pt, accessed on February 2002. Portugal.

[Szy02] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming.* Addison-Wesley Pub. Co., Essex, England, 2^{nd} edition, 2002. ISBN 0-201-74572-0.

[The92] The RAISE Language Group. *The RAISE Specification Language.* BCS Practitioner Series. Prentice-Hall International, London, Great Britain, 1992. ISBN 0-13-752833-7.

[TP96] Daniel Tkach and Richard Puttick. *Object Technology in Application Development.* Addison-Wesley Pub. Co., New York, United States of America, 2^{nd} edition, 1996. ISBN 0-201-49833-2.

[UU01] U.S. Geological Survey and U.S. Department of Interior. The National Map: Topographic Mapping for the 21^{st} Century. Technical report, Office of the Associate Director for Geography, U.S. Geological Survey, Reston, Va., United States of America, November 2001. Cooperative Topographic Mapping Program.

[Vel98] Daan Velthausz. *Cost-Effective Network-Based Multimedia Information Retrieval.* Ph.D. thesis, University of Twente, Enschede, The Netherlands, November 1998. Telematica Instituut Fundamental Research Series, No. 003.

[VFPQS99] Chris Vissers, Luís Ferreira Pires, Dick A. C. Quartel and Marten J. van Sinderen. *The architectural design of distributed systems: Reader for the Design of Telematic Systems.* University of Twente, Enschede, The Netherlands, January 1999.

[Vli00] Hans van Vliet. *Software Engineering: Principles and Practice*. John Wiley & Sons Ltd., West Sussex, England, 2^{nd} edition, 2000. ISBN 0-471-97508-7.

[W3C99] *Resource Description Framework (RDF) Model and Syntax Specification*. W3C recommendation, World Wide Web Consortium, Cambridge, MA, February 1999. Editors: Ora Lassila and Ralph R. Swick.

[W3C01a] *Web Services Description Language (WSDL) 1.1*. W3C recommendation, World Wide Web Consortium, Cambridge, MA, March 2001. Editors: Erik Christensen, Francisco Curbera, Greg Meredith and Sanjiva Weerawarana.

[W3C01b] *XML Schema Part 1: Structures*. W3c recommendation, World Wide Web Consortium, Cambridge, MA, May 2001. Editors: Henry S. Thompson, David Beech, Murray Maloney and Noah Mendelsohn.

[W3C01c] *XML Schema Part 2: Datatypes*. W3C recommendation, World Wide Web Consortium, Cambridge, MA, May 2001. Editors: Paul V. Biron and Ashok Malhotra.

[W3C03] 'The World Wide Web consortium.' www.w3.org, accessed on November 2003.

[WD96] Jim Woodcock and Jim Davies. *Using Z: Specification, Refinement and Proof*. Prentice-Hall International Series in Computer Science. Prentice-Hall International, Great Britain, 1996. ISBN 0-13-948472-8.

[Weg98] Marten K. de Weger. *Structuring of Business Processes: An architectural approach to system development and its application to business processes*. Ph.D. thesis, University of Twente, Enschede, The Netherlands, January 1998. Ph.D. Thesis, no. 98-17.

[Wil02] A. Denise Williams (editor). *Proceedings of the Sixth International Enterprise Distributed Object Computing Conference EDOC 2002*. Ecole Polytechnique Fédeérale de Laussane (EPFL), IEEE Computer Society, Lausanne, Switzerland, September 2002. ISBN 0-7695-1742-0.

[WJK99] Michael Wooldridge, Nicholas R. Jennings and David Kinny. 'A methodology for agent-oriented analysis and design.' In *Proceedings of the 3^{rd} International Conference on Autonomous Agents (Agents'99)*, edited by Oren Etzioni, Jörg P. Müller and Jeffrey M. Bradshaw, pages 69–76. ACM Press, Seattle, WA, USA, 1999.

[WKRG99] Brian Warboys, Peter Kawelek, Ian Robertson and Robert Greenwood. *Business Information Systems: a Process Approach*. Information Systems Series. McGraw Hill, Great Britain, 1999. ISBN 0-07-709464-6.

[Wor95] Michael F. Worboys. *GIS: A Computing Perspective*. Taylor & Francis Ltd., London, United Kingdom, 1995. ISBN 0-7484-0065-6.

[Zei99] Michael Zeiler. *Modelling Our World: The ESRI Guide to Geodatabase Design*. Environmental Systems Research Institute, Inc., Redlans, California, 1999. ISBN 1-879102-62-5.

Index

Summary

This thesis presents a method for the development of distributed geo-information systems. The method is organised around the design principles of modularity, reuse and replaceability. The method enables the modelling of both behavioural and informational aspects of geo-information systems in an integrated way.

This thesis introduces the concept the Geo-information Service Infrastructure (GSI). The GSI concept adheres to the characteristics and needs of modern geo-information processing. The GSI builds on the existing principles for data sharing of the Geospatial Data Infrastructure concept. The term GSI is used to refer to a type of geo-information provision system from which specialised information products and services can be obtained by exploiting the elementary services (resources, processes and data) of a set of collaborating geo-service providers. This thesis presents a supporting architecture for the deployment of GSI services.

An important idea underlying a GSI system is that services available in an information infrastructure should be composable. The method presented in this thesis is used to describe these elementary services and their interfaces, such that they can be accessed, combined and managed to create compound required services. These later services are defined to handle elaborated and specialised geo-processing tasks.

Services are specified according to two perspectives namely the external perspective and the internal perspective. The external perspective specifies the observable behaviour of a service. This specification is used by designers for the development of the internal perspective and for the assessment of the resulting internal service specification. A service specification according to the external perspective also provides potential users with a description of the service functionality and its interfacing mechanisms.

The internal perspective describes internal structure of a required service in terms of compositions of simpler or more elementary services. This internal structure is defined in terms of architectural elements. Three types of architectural elements are distinguished, viz. data elements, processing elements and connecting elements. The *data elements* represent the information that is used, manipulated and/or generated by the system. The *process elements* represent the geo-processing capabilities of

the system, which can perform transformations on data elements. The *connecting elements* or *mediators* coordinate the interactions between the other architectural elements, and provide an interface to the service user. All specifications, according to both the external and the internal perspectives, are defined using ISDL (Interaction Systems Design Language) concepts.

At the centre of a GSI system lies the repository service. The repository allows to organise the creation, updating, validation, accessing and sharing of service models and service instances. We emphasise the use of models as the mechanism to disclose information about services and to design more elaborated services out of combinations of existing elementary services. The repository is defined according to a metamodel on which all service models are based. The metamodel provides a rigourous abstract syntax for defining models. The metamodel is used here to define a set of design concepts and their relationships, which one can use to produce models according to the GSI specific objectives.

This thesis introduces and motivates the so-called mediator pattern to be used to structure compositions of architectural elements. This results in the organisation of a set of services into a behaviour definition that has a single coordinating element. One of the benefits of this approach is that it makes the service realisation accountable for the user. Another benefit is that it facilitates the use of workflow languages to implement the mediator behaviour, which choreographs the use of third-party services.

Samenvatting

Dit proefschrift presenteert een methode voor het ontwikkelen van gedistribueerde geo-informatie systemen. Deze methode is gebaseerd op ontwerpprincipes zoals modulariteit, hergebruik en onderhoudbaarheid. De methode ondersteunt het gecombineerd modelleren van zowel gedrags- als informatieaspecten van geo-informatie systemen

Dit proefschrift introduceert het begrip 'Geo-information Service Infrastructure' (GSI). Het GSI begrip sluit aan op de eigenschappen van en de eisen aan geavanceerde geo-informatie verwerking. GSI steunt op bestaande principes voor 'data sharing' zoals toegepast in een traditionele 'Geospatial Data Infrastructure'. De term GSI wordt gebruikt om een geo-informatie dienstensysteem aan te duiden waarmee gespecialiseerde informatieproducten en -diensten verkregen kunnen worden door elementaire diensten (middelen, processen en data) van een verzameling samenwerkende geo-dienstenaanbieders te exploiteren. Dit proefschrift presenteert een ondersteunde architectuur voor het uitrollen van GSI diensten.

Een funderende notie voor een GSI systeem is dat beschikbare diensten van een informatie infrastructuur compositioneel moeten zijn. De methode gepresenteerd in dit proefschrift is gebruikt om deze elementaire diensten en hun interfaces te beschrijven, zodat ze toegankelijk zijn, en gecombineerd en beheerd kunnen worden voor het samenstellen van de vereiste diensten. De resulterende diensten kunnen complexere en gespecialiseerde geo-verwerkingstaken ondersteunen.

Diensten worden gespecificeerd volgens twee gezichtspunten, namelijk het externe en het interne gezichtspunt. Het externe gezichtspunt definieert het observeerbare gedrag van een dienst. Deze specificatie wordt gebruikt door ontwerpers als uitgangspunt voor het ontwikkelen van het interne gezichtspunt, en voor de evaluatie van de resulterende dienstspecificatie volgens het interne gezichtspunt. Een dienstspecificatie volgens het externe gezichtspunt voorziet tevens de potentiële gebruikers van de beschrijving van de functionaliteit en de interface mechanismen van de dienst.

Het interne gezichtspunt beschrijft een interne structuur van de vereiste dienst als een samenstelling van eenvoudiger en meer elementaire diensten. Deze interne structuur wordt gedefinieerd in termen van architecturale elementen. Drie soorten architecturale elementen worden onderscheiden, namelijk data elementen, verwerkingselementen

en verbindingselementen. Data elementen representeren informatie die gebruikt, gemanipuleerd en/of gegenereerd wordt door het systeem. Verwerkingselementen representeren de geo-verwerkingsmogelijkheden van het systeem, waarmee transformaties uitgevoerd kunnen worden op data elementen. Verbindingselementen of 'mediators' coördineren de interacties tussen de andere architecturale elementen en bieden een interface aan de dienstgebruiker. Alle specificaties, volgens zowel het externe als interne gezichtspunt, worden gedefinieerd met gebruik van de specificatietaal ISDL ('Interaction Systems Design Language').

De kern van een GSI systeem wordt gevormd door de 'repository' dienst. Deze dienst ondersteunt het creëren, verversen, valideren, toegankelijk maken en delen van dienstmodellen en -instanties. Wij benadrukken het gebruik van modellen als mechanismen om informatie over diensten beschikbaar te stellen en om complexere diensten samen te stellen uit combinaties van bestaande elementaire diensten. De 'repository' is gedefinieerd op basis van een metamodel waarop tevens alle dienstmodellen zijn gebaseerd. Dit metamodel biedt een precieze abstracte syntaxis voor het definiëren van modellen. Het metamodel wordt hier gebruikt om een verzameling ontwerpconcepten en hun relaties te definiëren, zodat deze gebruikt kunnen worden voor het produceren van modellen voor GSI-specifieke doeleinden.

Dit proefschrift introduceert en motiveert het zogenaamde 'mediator' patroon, dat gebruikt wordt voor het structureren van composities van architecturale elementen. Dit resulteert in een structuur waarin een verzameling diensten worden gecoördineerd door een enkel coördinerend element. Een van de voordelen van deze benadering is dat de verantwoording aan de gebruiker ten aanzien van de dienstrealisatie wordt gegeven via een enkel element. Een ander voordeel van deze benadering is dat 'workflow' talen op een natuurlijke manier toegepast kunnen worden voor het implementeren van het 'mediator' gedrag dat het gebruik van diensten van derden regisseert.

Curriculum Vitae

Javier Marcelino Morales Guarin was born on the 16^{th} January, 1968, in Bogotá, Colombia. In 1992 he obtained his bachelor's degree in forestry engineering from the Distrital University 'Francisco José de Caldas' in Bogotá, Colombia. In 1998 he obtained his Master of Science degree in Geo-information Science from the International Institute for Geo-Information Science and Earth Observation in Enschede, the Netherlands. He was part-time lecturer at the faculty of civil engineering of the University of Santo Tomás in Bogotá. He has worked as a consultant in the areas of GIS and information technology, and he has also worked for more than 14 years in the geo-information industry with the national mapping organisation of Colombia in the areas of data acquisition, geographic information systems, map production and geo-information infrastructures. He is also member of the Architecture Group of the Telematics Services and Systems cluster of the faculty of Computer Science at the University of Twente. His present research interests are in design methods, architectures for distributed (geo-information) systems and spatial data infrastructures.

ITC Dissertations

[1] **Akinyede, Joseph O.**, 1990, *Highway Cost Modelling and Route Selection Using a Geotechnical Information System*, Delft University of Technology.

[2] **Pan, Ping He**, 1990, *A Spatial Structure Theory in Machine Vision and Applications to Structural and Textural Analysis of Remotely Sensed Images*, University of Twente, 90-9003757-8.

[3] **Bocco Verdinelli, Gerardo H. R.**, 1990, *Gully Erosion Analysis Using Remote Sensing and Geographic Information Systems: A Case Study in Central Mexico*, Universiteit van Amsterdam.

[4] **Sharif, Massoud**, 1991, *Composite Sampling Optimization for DTM in the Context of GIS*, Wageningen Agricultural University.

[5] **Drummond, Jane E.**, 1991, *Determining and Processing Quality Parameters in Geographic Information Systems*, University of Newcastle.

[6] **Groten, Susanne**, 1991, *Satellite Monitoring of Agro-ecosystems in the Sahel*, Westfälische Wilhelms-Universität.

[7] **Sharifi, Ali**, 1991, *Development of an Appropriate Resource Information System to Support Agricultural Management at Farm Enterprise Level*, Wageningen Agricultural University, 90-6164-074-1.

[8] **van der Zee, Dick**, 1991, *Recreation Studied from Above: Air Photo Interpretation as Input into Land Evaluation for Recreation*, Wageningen Agricultural University, 90-6164-075-X.

[9] **Mannaerts, Chris**, 1991, *Assessment of the Transferability of Laboratory Rainfall-runoff and Rainfall—Soil Loss Relationships to Field and Catchment Scales: A Study in the Cape Verde Islands*, University of Ghent, 90-6164-085-7.

[10] **Wang, Ze Shen**, 1991, *An Expert System for Cartographic Symbol Design*, Utrecht University, 90-3930-333-9.

[11] **Zhou, Yunxuan**, 1991, *Application of Radon Transforms to the Processing of Airborne Geophysical Data*, Delft University of Technology, 90-6164-081-4.

[12] **de Zuviría, Martín**, 1992, *Mapping Agro-topoclimates by Integrating Topographic, Meteorological and Land Ecological Data in a Geographic Information System: A Case Study of the Lom Sak Area, North Central Thailand*, Universiteit van Amsterdam, 90-6164-077-6.

[13] **van Westen, Cees J.**, 1993, *Application of Geographic Information Systems to Landslide Hazard Zonation*, Delft University of Technology, 90-6164-078-4.

[14] **Shi, Wenzhong**, 1994, *Modelling Positional and Thematic Uncertainties in Integration of Remote Sensing and Geographic Information Systems*, Universität Osnabrück, 90-6164-099-7.

[15] **Javelosa, R.**, 1994, *Active Quaternary Environments in the Philippine Mobile Belt*, Utrecht University, 90-6164-086-5.

[16] **Lo, King-Chang**, 1994, *High Quality Automatic DEM, Digital Elevation Model Generation from Multiple Imagery*, University of Twente, 90-9006-526-1.

[17] **Wokabi, S. M.**, 1994, *Quantified Land Evaluation for Maize Yield Gap Analysis at Three Sites on the Eastern Slope of Mt. Kenya*, University Ghent, 90-6164-102-0.

[18] **Rodríguez Parisca, O. S.**, 1995, *Land Use Conflicts and Planning Strategies in Urban Fringes: A Case Study of Western Caracas, Venezuela*, University of Ghent.

[19] **van der Meer, Freek D.**, 1995, *Imaging Spectrometry & the Ronda Peridotites*, Wageningen Agricultural University, 90-5485-385-9.

[20] **Kufoniyi, Olajide**, 1995, *Spatial Coincidence: Automated Database Updating and Data Consistency in Vector GIS*, Wageningen Agricultural University, 90-6164-105-5.

[21] **Zambezi, P.**, 1995, *Geochemistry of the Nkombwa Hill Carbonatite Complex of Isoka District, North-east Zambia, with Special Emphasis on Economic Minerals*, Vrije Universiteit Amsterdam.

[22] **Woldai, Tsehaie**, 1995, *The Application of Remote Sensing to the Study of the Geology and Structure of the Carboniferous in the Calañas Area, Pyrite Belt, South-west Spain*, Open University, United Kingdom.

[23] **Verweij, Pita A.**, 1995, *Spatial and Temporal Modelling of Vegetation Patterns: Burning and Grazing in the Páramo of Los Nevados National Park, Colombia*, Universiteit van Amsterdam, 90-6164-109-8.

[24] **Pohl, Christine**, 1996, *Geometric Aspects of Multisensor Image Fusion for Topographic Map Updating in the Humid Tropics*, Universität Hannover, 90-6164-121-7.

[25] **Bin, Jiang**, 1996, *Fuzzy Overlay Analysis and Visualization in Geographic Information Systemes*, Utrecht University, 90-6266-128-9.

[26] **Metternicht, Graciela I.**, 1996, *Detecting and Monitoring Land Degradation Features and Processes in the Cochabamba Valleys, Bolivia. A Synergistic Approach*, University of Ghent, 90-6164-118-7.

[27] **Chu Thai Hoanh**, 1996, *Development of a Computerized Aid to Integrated Land Use Planning (CAILUP) at Regional Level in Irrigated Areas: A Case Study for the Quan Lo Phung Hiep region in the Mekong Delta, Vietnam*, Wageningen Agricultural University, 90-6164-120-9.

[28] **Roshannejad, A.**, 1996, *The Management of Spatio-Temporal Data in a National Geographic Information System*, University of Twente, 90-9009-284-6.

[29] **Terlien, Mark T. J.**, 1996, *Modelling Spatial and Temporal Variations in Rainfall-triggered Landslides: The Integration of Hydrologic Models, Slope Stability Models and GIS for the Hazard Zonation of Rainfall-triggered Landslides with Examples from Manizales, Colombia*, Utrecht University, 90-6164-115-2.

[30] **Mahavir, J.**, 1996, *Modelling Settlement Patterns for Metropolitan Regions: Inputs from Remote Sensing*, Utrecht University, 90-6164-117-9.

[31] **Al-Amir, Sahar**, 1996, *Modern Spatial Planning Practice as Supported by the Multi-applicable Tools of Remote Sensing and GIS: The Syrian Case*, Utrecht University, 90-6164-116-0.

[32] **Pilouk, M.**, 1996, *Integrated Modelling for 3D GIS*, University of Twente, 90-6164-122-5.

[33] **Duan, Zengshan**, 1996, *Optimization Modelling of a River-Aquifer System with Technical Interventions: A Case Study for the Huangshui River and the Coastal Aquifer, Shandong, China*, Vrije Universiteit Amsterdam, 90-6164-123-3.

[34] **de Man, W. H. E.**, 1996, *Surveys: Informatie als Norm: Een Verkenning van de Institutionalisering van Dorp-surveys in Thailand en op de Filippijnen*, University of Twente, 90-9009-775-9.

[35] **Vekerdy, Zoltan**, 1996, *GIS-based Hydrological Modelling of Alluvial Regions: Using the Example of the Kisafld, Hungary*, Lorand Eotvos University of Sciences, 90-6164-119-5.

[36] **Gomes Pereira, Luisa M.**, 1996, *A Robust and Adaptive Matching Procedure for Automatic Modelling of Terrain Relief*, Delft University of Technology, 90-407-1385-5.

[37] **Fandiño Lozano, M. T.**, 1996, *A Framework of Ecological Evaluation oriented at the Establishment and Management of Protected Areas: A Case Study of the Santuario de Iguaque, Colombia*, Universiteit van Amsterdam, 90-6164-129-2.

[38] **Toxopeus, Bert**, 1996, *ISM: An Interactive Spatial and Temporal Modelling System as a Tool in Ecosystem Management: With Two Case Studies: Cibodas Biosphere Reserve, West Java Indonesia: Amboseli Biosphere Reserve, Kajiado District, Central Southern Kenya*, Universiteit van Amsterdam, 90-6164-126-8.

[39] **Wang, Yiman**, 1997, *Satellite SAR Imagery for Topographic Mapping of Tidal Flat Areas in the Dutch Wadden Sea*, Universiteit van Amsterdam, 90-6164-131-4.

[40] **Saldana Lopez, Asun**, 1997, *Complexity of Soils and Soilscape Patterns on the Southern Slopes of the Ayllon Range, Central Spain: a GIS Assisted Modelling Approach*, Universiteit van Amsterdam, 90-6164-133-0.

[41] **Ceccarelli, T.**, 1997, *Towards a Planning Support System for Communal Areas in the Zambezi Valley, Zimbabwe; A Multi-criteria Evaluation Linking Farm Household Analysis, Land Evaluation and Geographic Information Systems*, Utrecht University, 90-6164-135-7.

[42] **Peng, Wanning**, 1997, *Automated Generalization in GIS*, Wageningen Agricultural University, 90-6164-134-9.

[43] **Mendoza Lawas, M. C.**, 1997, *The Resource Users' Knowledge, the Neglected Input in Land Resource Management: The Case of the Kankanaey Farmers in Benguet, Philippines*, Utrecht University, 90-6164-137-3.

[44] **Bijker, Wietske**, 1997, *Radar for Rain Forest: A Monitoring System for Land Cover Change in the Colombian Amazon*, Wageningen Agricultural University, 90-6164-139-X.

[45] **Farshad, Abbas**, 1997, *Analysis of Integrated Soil and Water Management Practices within Different Agricultural Systems under Semi-arid Conditions of Iran and Evaluation of their Sustainability*, University of Ghent, 90-6164-142-X.

[46] **Orlic, B.**, 1997, *Predicting Subsurface Conditions for Geotechnical Modelling*, Delft University of Technology, 90-6164-140-3.

[47] **Bishr, Yaser**, 1997, *Semantic Aspects of Interoperable GIS*, Wageningen Agricultural University, 90-6164-141-1.

[48] **Zhang, Xiangmin**, 1998, *Coal fires in Northwest China: Detection, Monitoring and Prediction Using Remote Sensing Data*, Delft University of Technology, 90-6164-144-6.

[49] **Gens, Rudiger**, 1998, *Quality Assessment of SAR Interferometric Data*, University of Hannover, 90-6164-155-1.

[50] **Turkstra, Jan**, 1998, *Urban Development and Geographical Information: Spatial and Temporal Patterns of Urban Development and Land Values Using Integrated Geo-data, Villaviciencio, Colombia*, Utrecht University, 90-6164-147-0.

[51] **Cassells, Craig James Steven**, 1998, *Thermal Modelling of Underground Coal Fires in Northern China*, University of Dundee.

[52] **Naseri, M. Y.**, 1998, *Monitoring Soil Salinization, Iran*, Ghent University, 90-6164-195-0.

[53] **Gorte, Ben G. H.**, 1998, *Probabilistic Segmentation of Remotely Sensed Images*, Wageningen Agricultural University, 90-6164-157-8.

[54] **Ayenew, Tenalem**, 1998, *The Hydrological System of the Lake District Basin, Central Main Ethiopian Rift*, Universiteit van Amsterdam, 90-6164-158-6.

[55] **Wang, Donggen**, 1998, *Conjoint Approaches to Developing Activity-Based Models*, Technical University of Eindhoven, 90-6864-551-7.

[56] **Bastidas de Calderon, María**, 1998, *Environmental Fragility and Vulnerability of Amazonian Landscapes and Ecosystems in the Middle Orinoco River Basin, Venezuela*, University of Ghent.

[57] **Moameni, A.**, 1999, *Soil Quality Changes under Long-term Wheat Cultivation in the Marvdasht Plain, South-central Iran*, University of Ghent.

[58] **van Groenigen, J.W.**, 1999, *Constrained Optimisation of Spatial Sampling: A Geostatistical Approach*, Wageningen Agricultural University, 90-6164-156-X.

[59] **Cheng, Tao**, 1999, *A Process-oriented Data Model for Fuzzy Spatial Objects*, Wageningen Agricultural University, 90-6164-164-0.

[60] **Wolski, Piotr**, 1999, *Application of Reservoir Modelling to Hydrotopes Identified by Remote Sensing*, Vrije Universiteit Amsterdam, 90-6164-165-9.

[61] **Acharya, B.**, 1999, *Forest Biodiversity Assessment: A Spatial Analysis of Tree Species Diversity in Nepal*, Leiden University, 90-6164-168-3.

[62] **Abkar, Ali Akbar**, 1999, *Likelihood-based Segmentation and Classification of Remotely Sensed Images*, University of Twente, 90-6164-169-1.

[63] **Yanuariadi, Tetra**, 1999, *Sustainable Land Allocation: GIS-based Decision Support for Industrial Forest Plantation Development in Indonesia*, Wageningen University, 90-5808-082-X.

[64] **Abu Bakr, Mohamed**, 1999, *An Integrated Agro-Economic and Agro-Ecological Framework for Land Use Planning and Policy Analysis*, Wageningen University, 90-6164-170-5.

[65] **Eleveld, Marieke A.**, 1999, *Exploring Coastal Morphodynamics of Ameland (The Netherlands) with Remote Sensing Monitoring Techniques and Dynamic Modelling in GIS*, Universiteit van Amsterdam, 90-6461-166-7.

[66] **Hong, Yang**, 1999, *Imaging Spectrometry for Hydrocarbon Microseepage*, Delft University of Technology, 90-6164-172-1.

[67] **Mainam, Félix**, 1999, *Modelling Soil Erodibility in the Semiarid Zone of Cameroon*, University of Ghent, 90-6164-179-9.

[68] **Bakr, Mahmoud I.**, 2000, *A Stochastic Inverse-Management Approach to Groundwater Quality*, Delft University of Technology, 90-6164-176-4.

[69] **Zlatanova, Siyka**, 2000, *3D GIS for Urban Development*, Graz University of Technology, 90-6164-178-0.

[70] **Ottichilo, Wilber K.**, 2000, *Wildlife Dynamics: An Analysis of Change in the Masai Mara Ecosystem*, Wageningen University, 90-5808-197-4.

[71] **Kaymakci, Nuri**, 2000, *Tectono-stratigraphical Evolution of the Cankori Basin (Central Anatolia, Turkey)*, Utrecht University, 90-6164-181-0.

[72] **Gonzalez, Rhodora**, 2000, *Platforms and Terraces: Bridging Participation and GIS in Joint-learning for Watershed Management with the Ifugaos of the Philippines*, Wageningen University, 90-5808-246-6.

[73] **Schetselaar, Ernst**, 2000, *Integrated Analyses of Granite-gneiss Terrain from Field and Multisource Remotely Sensed Data. A Case Study from the Canadian Shield*, University of Delft, 90-6164-180-2.

[74] **Mesgari, M. Saadi**, 2000, *Topological Cell-Tuple Structure for Three-Dimensional Spatial Data*, University of Twente, 90-3651-511-4.

[75] **de Bie, Cees A. J. M.**, 2000, *Comparative Performance Analysis of Agro-Ecosystems*, Wageningen University, 90-5808-253-9.

[76] **Khaemba, Wilson M.**, 2000, *Spatial Statistics for Natural Resource Management*, Wageningen University, 90-5808-280-6.

[77] **Shrestha, Dhruba**, 2000, *Aspects of Erosion and Sedimentation in the Nepalese Himalaya: Highland-lowland Relations*, Ghent University, 90-6164-189-6.

[78] **Asadi Haroni, Hooshang**, 2000, *The Zarshuran Gold Deposit Model Applied in a Mineral Exploration GIS in Iran*, Delft University of Technology, 90-6164-185-3.

[79] **Raza, Ale**, 2001, *Object-Oriented Temporal GIS for Urban Applications*, University of Twente, 90-3651-540-8.

[80] **Farah, Hussein O.**, 2001, *Estimation of Regional Evaporation under Different Weather Conditions from Satellite and Meteorological Data. A Case Study in the Naivasha Basin, Kenya*, Wageningen University, 90-5808-331-4.

[81] **Zheng, Ding**, 2001, *A Neuro-Fuzzy Approach to Linguistic Knowledge Acquisition and Assessment in Spatial Decision Making*, University of Vechta, 90-6164-190-X.

[82] **Sahu, B. K.**, 2001, *Aeromagnetics of Continental Areas Flanking the Indian Ocean; with Implications for Geological Correlation and Gondwana Reassembly*, University of Capetown, South Africa.

[83] **Alfestawi, Yahia Ahmed M.**, 2001, *The Structural, Paleogeographical and Hydrocarbon Systems Analysis of the Ghadamis and Murzuq Basins, West Libya, with Emphasis on Their Relation to the Intervening Al Qarqaf Arch*, Delft Technical University, 90-6164-198-5.

[84] **Liu, Xuehua**, 2001, *Mapping and Modelling the Habitat of Giant Pandas in Foping Nature Reserve, China*, Wageningen University, 90-5808-496-5.

[85] **Oindo, Boniface Oluoch**, 2001, *Spatial Patterns of Species Diversity in Kenya*, Wageningen University, 90-5808-495-7.

[86] **Carranza, Emmanuel John M.**, 2002, *Geologically-constrained Mineral Potential Mapping: Examples from the Philippines*, Technical University of Delft, 90-6164-203-5.

[87] **Rugege, Denis**, 2002, *Regional Analysis of Maize-based Land Use Systems for Early Warning Applications*, Wageningen University, 90-5808-584-8.

[88] **Liu, Yaolin**, 2002, *Categorical Database Generalization in GIS*, Wageningen University, 90-5808-648-8.

[89] **Ogao, Patrick**, 2002, *Scientific Visualization*, Utrecht University, 90-6164-206-X.

[90] **Abadi, Abdulbaset Musbah**, 2002, *Tectonics of the Sirt Basin: Inferences from Tectonic Subsidence Analysis, Stress Inversion and Gravity Modeling*, Vrije Universiteit Amsterdam, 90-6164-205-1.

[91] **Geneletti, Davide**, 2002, *Ecological Evaluation for Environmental Impact Assessment*, Vrije Universiteit Amsterdam, 90-6809-337-1.

[92] **Sedogo, Laurent D.**, 2002, *Integration of Local Participatory and Regional Planning for Resources Management Using Remote Sensing and GIS*, Wageningen University, 90-5808-751-4.

[93] **Montoya, Ana Lorena**, 2002, *Urban Disaster Management: A Case Study of Earthquake Risk Assesment in Cartago, Costa Rica*, Utrecht University, 90-6164-2086.

[94] **Mobin-ud Din, Ahmad**, 2002, *Estimation of Net Groundwater Use in Irrigated River Basins Using Geo-information Techniques: A Case Study in Rechna Doab, Pakistan*, Wageningen University, 90-5808-761-1.

[95] **Said, Mohammed Yahya**, 2003, *Multiscale Perspectives of Species Richness in East Africa*, Wageningen University, 90-5808-794-8.

[96] **Schmidt, Karen S.**, 2003, *Hyperspectral Remote Sensing of Vegetation Species Distribution in a Saltmarsh*, Wageningen University, 90-5808-830-8.

[97] **López Binnqüist, Citlalli**, 2003, *The Endurance of Mexican Amate Paper: Exploring Additional dimensions to the Sustainable Development Concept*, University of Twente, 90-3651-900-4.

[98] **Huang, Zhengdong**, 2003, *Data Integration for Urban Transport Planning*, Utrecht University, 90-6164-211-6.

[99] **Cheng, Jianquan**, 2003, *Modelling Spatial and Temporal Urban Growth*, Utrecht University, 90-6164-212-4.

[100] **Campos dos Santos, José Laurindo**, 2003, *A Biodiversity Information System in an Open Data-Metadatabase Architecture*, University of Twente, 9061642140.

[101] **Hengl, Tomislav**, 2003, *Pedometric Mapping: Bridging the Gaps Between Conventional and Pedometric Approaches*, Wageningen University.

[102] **Barrera Bassols, Narciso**, 2003, *Symbolism, Knowledge and management of Soil and Land Resources in Indigenous Communities: Ethnopedology at Global, Regional and Local Scales*, University of Ghent.

[103] **Zhan, Qingming**, 2003, *A Hierarchical Object-based Approach for Urban Land-use Classification from Remote Sensing Data*, Wageningen University, 90-5808-917-7.

[104] **Daag, Arturo Santos**, 2003, *Modelling the Erosion of the Pyroclastic Flow Deposits and the Occurrences of Lahars at Mt. Pinatubo, Philipines*, Utrecht University, 90-6164-218-3.

[105] **Bacic, Ivan Luiz Zilli**, 2003, *Demand Driven Land Evaluation: With Case Studies in Santa Catarina, Brazil*, Wageningen University, 90-5808-902-9.

[106] **Murwira, Amon**, 2003, *Scale matters! A New Approach to Quantify Spatial Heterogeneity for Predicting the Distribution of Wildlife*, Wageningen University.

[107] **Mazvimavi, Dominic**, 2003, *Estimation of Flow Characteristics of Ungauged Catchments: A Case Study in Zimbabwe*, Wageningen University, 90-5808-950-9.

[108] **Tang, Xinming**, 2004, *Spatial Object Modeling in Fuzzy Topological Spaces: With Applications to Land Cover Change*, University of Twente, 90-6164-2205.

[109] **Kariuki, Patrick C.**, 2004, *Spectroscopy to measure the swelling potential of expansive soils*, University of Delft, 90-6164-221-3.