Resource Access in Generic Information Retrieval Systems

Eric D. Schabell

June 30, 2002

Preface

The following thesis is for the authors Masters Degree (or 'doctorandus') in Computer Science at the Free University in Amsterdam, The Netherlands. This thesis describes his search for a generic software architecture for use in generic information retrieval. This work was carried out entirely at the University of Nijmegen, under the supervision of Prof. Dr. Erik Proper and thesis mentor Dr. Ir. Hans de Bruin from the Free University of Amsterdam.

Acknowledgments

To finally be writing a Master's Thesis is a dream come true. The path taken has been long, involving numerous part-time jobs, various academic prerequisites and last but not least a large amount of my evening hours over the years. My dedication was not enough to get me through, there have been numerous helping hands along the way. Please bear with me as I mention the few that spring readily to mind and forgive me for the ones I have forgotten, you are not in any way less important.

To begin with I would like to thank my on site supervisor Prof. Dr. Erik Proper at the University of Nijmegen. He was always open to my ideas and patient with my questions. He has drawn me into information retrieval in a way that I never thought would have been possible and I look forward to the continuing journey.

My mentor at the Free University in Amsterdam, Dr. Ir. Hans de Bruin was also a pleasure to work with. From the beginning he was always supportive and encouraging. His extensive knowledge of the applicable research literature was an enormous source of help when I occasionally ran into a wall. His suggestions and tips always helped me to keep it on track.

I would also like to extend my thanks to the IRIS group as a whole for their support, cooperation, interviews, brainstorm sessions and coffee, there was never a lack of enthusiasm!

I would also like to mention a few organizations and people who have each in their own way contributed along the way. Starting with the Marine Corps, for showing me that studying is a privilege, that there are places in the world where education is unheard of and for giving me the discipline to never give up. To my parents who continually supported all my endeavors (I made it mom!) and my sister who helped in more ways than she will ever know. To the various Dutch employers who gave me room to work and study. To the Vu-clan, you have been the constant factor in my rather hectic study life and the biggest reason I chose to stay in the Netherlands, may we find ourselves always together. Finally, my girlfriend and later wife, who surely had the most hinder from all the time I spent in the books. Though she had to miss me the most she never complained, brought many a cup of coffee, sometimes a cookie and always her love.

Abstract

This thesis looks at the search for a generic information retrieval software architecture. I start with background information on the state of generic information retrieval, the organizations involved in this project and then explain the research goals that my thesis will attempt to reach.

The requirements analysis is done through the use of interviews due to a lack of existing systems to examine. By lack of existing systems, I am referring to the lack of existing information retrieval architectures that allow for the modular construction of generic information retrieval systems. The modular construction is important to facilitate the building of research prototypes and this is missing in the existing information retrieval architectural solutions, which tend to focus on single solution domains.

The requirements analysis resulted in the narrowing of my scope to resource access. An architectural reference model also grew out of the analysis and the next step was to expand this into a reference architecture. By using the requirements analysis as a starting point, three use cases were generated to use as a basis for creation of the reference architecture.

A domain analysis using the three use cases made it possible to take a look at the possible features and a feature diagram was created to show their relationships. The resulting reference architecture was tested in a proof of concept by mapping these features to stubs and plug-ins using the use case map as a notational tool.

The final software architecture is presented with stubs and various plug-ins used to denote the options used to fulfill both the functional and non-functional requirements that were set at the beginning of this project. This software architecture is explained and the conclusions are presented with a few suggestions for further development.

Contents

Chapter 1

Introduction

This chapter covers the introduction of my masters thesis. I start out with a background on the research problem, give some details about the Information Retrieval Information Systems group (IRIS), cover their global research strategy, present a list of goals that drive this thesis and close with a list of the definitions that apply to generic information retrieval research.

1.1 Background

The knowledge and information resources we need to conduct our activities in daily life, be it at work or at home, are increasingly available in some electronic form. The diversity of these resources are enormous: documents, people (by their contact addresses), document collections, objects or facts in databases and even entire applications. This "...still increasing, bombardment of (heterogeneous) information has led to a feeling of information overload."[Proper99]. The ability to search and more importantly find these heterogeneous networked resources is becoming a very complex issue in todays Internet environment.

To support users in their quest for the right resources the Information Retrieval and Information Systems (IRIS) group at the Catholic University in Nijmegen (KUN) has started a research project "Profile based retrieval of networked information resources (PRONIR)" in an attempt to "...develop a theory and demonstrate its validity by means of a prototype system, for profile based retrieval of heterogeneous networked resources."[Proper01]. The PRONIR project will be covered in more detail in section 1.3.

1.2 The IRIS group

The Information Retrieval and Information Systems (IRIS) group "...deals with the development of formal methods for the specification, design, analysis, exploration and validation of information systems."[IRISweb].

The group is a sub-group of the Computer Sciences faculty at the University of Nijmegen (KUN), Netherlands. The IRIS group is focused on the following general research goals:

- Description techniques, varying from conventional data modeling techniques to representation techniques for describing document contents.
- Systematic design of a structured data description, with a special emphasis on validation and deriving from such a specification a system/language to communicate about the application domain on a formal basis (propositional language, inference mechanism). This may contribute to the methodology within that data domain.
- Via detailed (off-line) analysis (data mining, virtual lab) prospecting of semantical properties and techniques to generate from this description an operational system (database).
- Fundamental research to communication (semiotic aspects) and knowledge.
- Techniques for automatic extraction of document contents.
- Data retrieval techniques.
- An architecture and approach in terms of a long term vision.

The last subject area will be the focus of my thesis, starting with a generic information retrieval software architecture and narrowing the scope to deal with the supply of information resources.

1.3 The PRONIR project

The PRONIR research project has two distinct directions of focus, profile based information retrieval and uniform resource access. These two areas will be further researched by PhD. students working within the IRIS group.

1.3.1 Profile based retrieval

Here the focus is on the ability of an information retrieval system to tune the set of retrieved resources to the specific information needs of a searcher. A pivotal role is played by a searchers profile. This profile can contain such aspects as the defaults that a searcher may harbor with respect to the usage of search terms. For example, think of a search based on “surfing”, is it “web surfing” or “wave surfing” that the searcher is referring to? Another aspect is the searcher’s aim for retrieving the resources, such as for reference purposes, subject orientation, an in-depth study, contact information, etc. Specific research goals with regards to this area of focus are:

- The definition (both syntax and semantics) of a profile specification language.
- Development of strategies and algorithms to define user, task and role specific profiles.
- Development of logic based information retrieval mechanisms which include the searcher's profile(s).

1.3.2 Uniform resource access

The search for a uniform way of modeling and characterizing heterogeneous knowledge resources is the subject of uniform resource access. It will mainly work on providing uniform views of different properties, such as meta-data of information resources and relationships between information resources. This project is not concerned with standardization of transfer protocols or storage formats. Here one would like to go beyond currently available meta-data standards and look at such aspects as supporting searchers in formulating their needs, requiring information retrieval systems to have a good understanding of the structure and semantics of the domains used to characterize the information resources. A search query should ideally be channeled based on the definition of a characterization domain and act accordingly, so in addition to the definition of characterization domains, it will be necessary to provide definitions of translations between these domains. Furthermore, within a collection of resources (a corpus) certain rules may be applicable with respect to the relevance of these resources to a searcher's needs. The goals with regards to this area of focus are:

- The definition of a (machine interpretable) language to define characterization domains.
- A transformation mechanism to provide translations between different characterization domains (when applicable).
- A way to provide access to constantly changing resource collections so that the characterization domains remain automatically up to date with regards to information in the generic information retrieval system.

1.4 Problem statement

To facilitate the KUN (and its research AIO's) in their information retrieval research there is a need for a flexible environment in which the building blocks are available to construct research prototypes with the minimum of software engineering effort, both in terms of defining architecture as well as programming. The concept of an architectural test bed and prototyping environment will be referred to as Generic Information Retrieval (GIR) in the rest of this thesis.

The KUN has already made a choice in terms of developing their prototype system as Free Software. This will therefore affect the reference architecture in that any implementation will also require the use of Free Software components.

1.5 Thesis goals

The following goals have been established for this thesis:

1. Determine the concepts that are involved with generic information retrieval.
2. Determine the requirements needed for the prototyping architecture.
3. Determine a reference architecture for the KUN's prototyping needs.
4. Attempt to provide a solution architecture (in the form of a Proof of Concept) for the interfacing between the characterization domains and the resource domains.

The first goal is reached by means of researching the available generic information retrieval literature. This goal was used as a starting point to become familiar with the research domain and to determine that the following goals had not yet been achieved.

1.6 Research structure

I began with a look at the available research in the field of generic information retrieval and other closely related areas. This was important in the sense that it provided an orientation as to the current state of generic information retrieval and provided a starting point for a deeper look at the needs of the IRIS group with regards to a generic information retrieval software architecture. I made a decision to present a larger overview called the search for the Holy Grail[1] and then focus on a subset of the whole to facilitate the time constraints placed on my thesis. A set of general requirements were established based on resource access, which applies to the Representation and Resource Domains. Chapter 2 will discuss the search for the Holy Grail and the supply side of generic information retrieval.

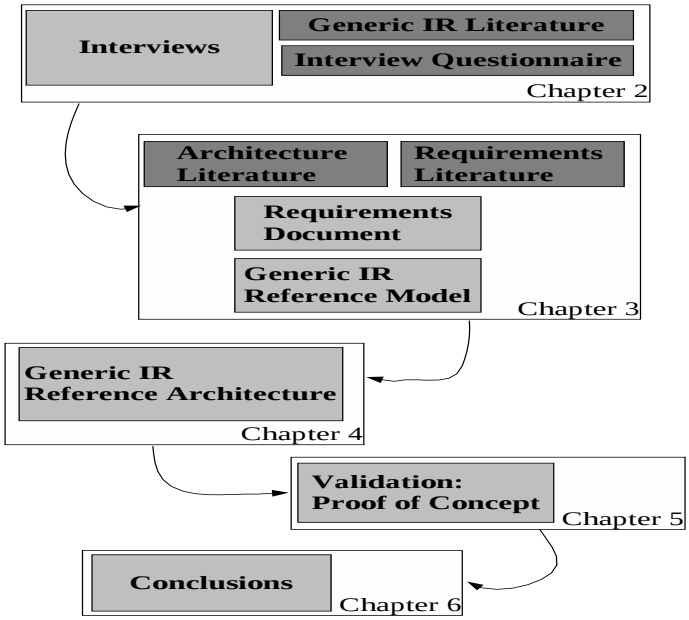
Next a basic questionnaire was compiled for distribution to the members of the IRIS staff that are involved in this area of research. Interviews were held to extract the requirements for resource access in a generic information retrieval software architecture. The requirements led to the almost parallel development of a reference model, which is

[1] Footnote to Holy Grail: The Holy Grail, a golden challis, was sought by the knights of the Round Table and led by the famous mythical King Arthur in old English folk lore. It was an unattainable object, which here is used to refer to the seemingly insurmountable task of finding a satisfactory software architecture for Generic Information Retrieval.

a model that provides "...a division of functionality together with data flow between the pieces."[Bass98].

The reference model was then converted into a reference architecture which maps the reference model onto software components and the data flows between these components.

The reference architecture was applied as a Proof of Concept, a mapping of the results to the defined part of the generic information retrieval system outlined in the requirements document. From this Proof of Concept it was then possible to validate and draw the conclusions that complete this thesis project. The following is a visual outline of the path followed in this thesis:



Chapter 2

Generic information retrieval

In this chapter I will start with a background on basic information retrieval. I will present a discussion of generic information retrieval as a search for the Holy Grail, providing the basic component layers involved and their basic interactions. I will then narrow this view down to describe resource access, which is a focus on the supply of information to a generic information retrieval system. Finally, I conclude with a look at the requirements obtained through interviews with the participants involved.

2.1 Information retrieval

Since the beginning of time we have been trying to retrieve information in one form or another to satisfy questions we have had. The basic idea is that information is stored someplace so that we can retrieve it without too much trouble. To retrieve this information there must be some sort of system that we can model to allow us to formulate our information desire and match this desire to an index of the stored information. The basic components involved in information retrieval are as follows:

- **formulation**
- **matching**
- **indexing**

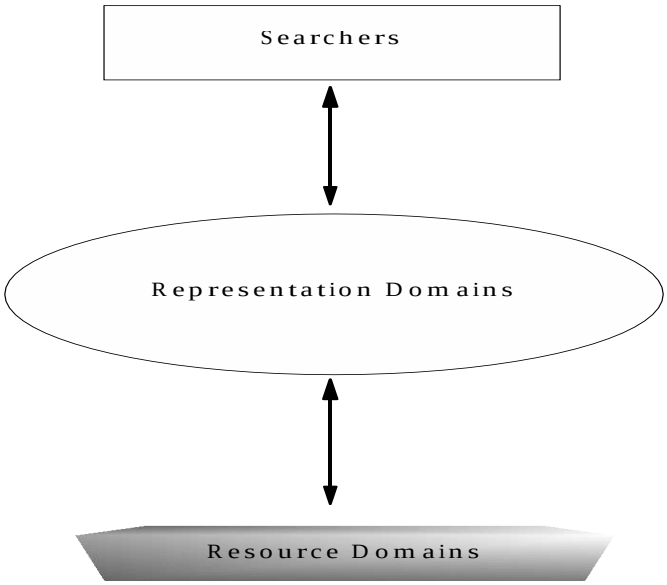
The process runs as follows, the user *formulates* an information need in the form of a request. This request is then passed into the system used to search the stored information. This system will have some sort of *index* applied to the stored information to allow for easy access and searching. The request will be applied to the *index*, this will result in the *matching* and retrieval (or reporting of request not available in the

stored information) of the desired information. This information is sent back to the user as an answer set to the original request.

This is a simple description of the process that will be referred to in the rest of this thesis as information retrieval. For more information and an interesting discussion on the Information Retrieval Paradigm, see [Bruza93].

2.2 The Holy Grail

The search for a generic information retrieval system can be compared to the search for the Holy Grail, as a search for the elusive and not easily discoverable. This search has begun for the IRIS group in the Netherlands. A generic information retrieval system has three distinct areas of operation that will be discussed from a top down view of the system; the Searchers, the Representation Domains and the Resource Domains. These layers each have their own area of focus and their own informational needs that can be grouped together when looking at the generic information retrieval environment. Below is a graphical representation of the three layers, for their concise definitions see section 7.1:



2.2.1 Searchers

The searchers layer represents the starting point of a request for information that will be handed down to the representation domains. This should be seen in the most generic sense, such that this can be a query, an application, an individual user or whatever is attempting to make a request for information from the next layer.

2.2.2 Representation domains

A representation domain describes the point where the searchers and the actual information are brought together. Information request are processed and passed down to the resource domains and replies from the resource domains are collected for processing before being presented back to the searchers layer. It is most likely that a request will be split into diverse sub-requests at this level and then sent down to different resource domains. This requires that the representation domain generates a collection of results and then processes this for presentation before passing it up to the searcher.

2.2.3 Resource domains

These are the collections of all resources within the system for a particular domain. For example, a specific Resource Domain could be defined to contain various document collections containing the information that a company uses to manage its daily business. These could be from the different departments such as purchasing, logistics and personnel offices. Each is a separate database but together they are seen as one collection in the defined Resource Domain. This is where the information is physically collected for a particular request from the representation domain. The results are passed to the representation domain for further processing.

2.3 Information supply

I will only be considering the supply side of generic information retrieval, that is the supply of new information to a system. This will require taking a look at the representation domain and part of the resource domain. Furthermore, the assumption will be made that document collections are the only collections allowed in our resource domains. This gives a set of attributes based on documents that are easily understood for further discussion and demonstration in this thesis.

2.4 Interviews

It was necessary to determine the requirements needed by the IRIS group for prototyping within a generic information retrieval system framework. The problem with this research is that it covers an area that is somewhere between architecture and software. This required a bit of flexibility on the part of any requirements analysis methodology that might be chosen. The result was a combination of software engineering, inquiry and architectural techniques. More can be found on the methods used in [Klein99, Bass98, Kzaman00, Potts94, Vliet93, IEEE98].

There was a lack of documentation of existing systems to study. This statement refers to the domain specific existing systems which do

not apply the concept of generic information retrieval. The goal here is to create a software architecture for prototyping within information retrieval systems. Generic information retrieval theory and research literature was used in combination with interviews will be used to examine for requirements. To keep the focus on generic information retrieval an explicit choice was made to deviate from the existing domain specific systems, which do not allow for easy experimentation.

Since there was no documentation of an existing generic system to study and no existing system code to examine, a choice was made for interviews with the IRIS group to obtain the requirement specifications. Admittedly, this is the least preferred method for ascertaining requirements, but due to the lack of physical or documented systems in this area of expertise there was little other choice. One can be comforted by the thought that the interviewed IRIS personnel are all experienced in the field of information retrieval and are currently participating in research activities pertaining to the area of generic information retrieval.

To prepare the subjects for the interview, a questionnaire was developed consisting of background information on generic information retrieval, the goals of the interview and a few questions that would guide the subjects in the right direction with regards to their own research experiences in this field to date. See Appendix 7.2 for the complete questionnaire.

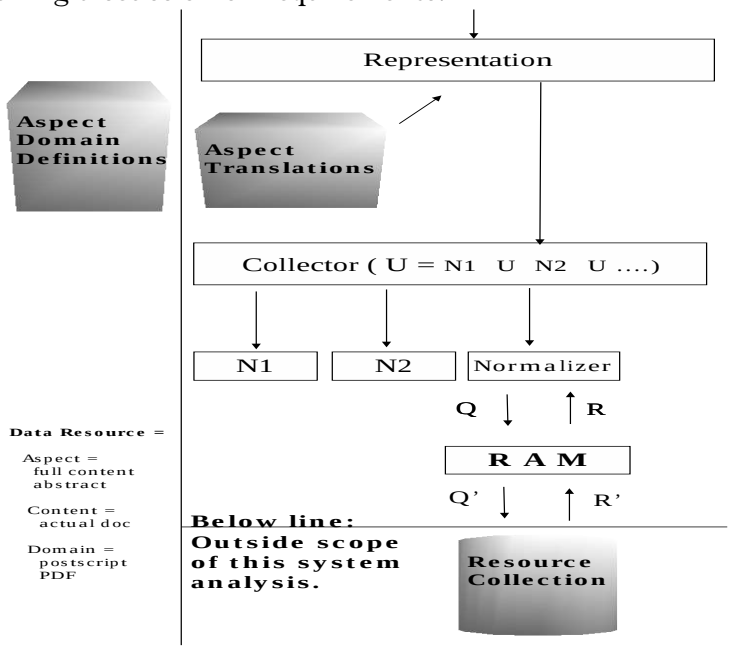
Chapter 3

Requirements analysis

In this chapter I will take a look at the requirements analysis and the requirements document that was delivered. I conclude with the presentation of an architectural reference model which emerged from the analysis.

3.1 Conceptual view

After conducting the interviews and many discussions on the subject of generic information retrieval with Prof. Dr. Erik Proper a requirements document was created. The document was constructed conform the IEEE requirements specifications standard [IEEE98] and can be found in Appendix 7.3. The following diagram will be the basis for the following discussion on requirements:



Refer to section 7.1 for detailed definitions of the listed components. The diagram details a generic information retrieval system at a conceptual level, providing both scope for this thesis and a means for communication with regards to the systems requirements. It is the first step to "...communicate clearly and unambiguously the architecture to all of the stake-holders who have an interest in it."[Bass98]. The diagram grew slowly from the interviews and discussions about the system. At the end of this chapter the reference model becomes the starting point from which a reference architecture will be constructed.

3.2 The requirements

The following section will present the requirements for the generic information retrieval system with regards to information supply. It will discuss the overall product view, the functional and the non-functional requirements.

3.2.1 Product view

The software architecture will be looked at with respect to the services that it will provide (or not provide) to the users, developers and researchers. The services to be provided can be split as follows:

- Add New Resource Collection.
- Modify Resource Collection Scheme Definition.
- Modify Resource Collection Population.
- Add New Aspect Domain Definition.
- Add Translation Definition.

As this is a generic architecture there are, in theory, no constraints on the types of objects that can be put into this system for retrieval (video, documents, pictures) and no constraints on the ways in which an object can be viewed. As we are still looking for this Holy Grail of generic information retrieval, this thesis will take a narrower view and will abide by the following general constraints:

- The only Information Resources to be considered will be document collections.
- Time for designing the Representation and Resource Domain software architecture is limited to 16 weeks.
- The basis for the software architecture will be a Free Software environment.

The entire generic information retrieval software architecture will be based on the concepts and components available in the Free Software

community. This enables third party involvement such as MSc students, programmers from interested software companies and the Free Software community in general. This means that eventual code and documentation will have to be made available to the Free Software community, either by maintaining it locally or turning it over to part of a Free Software project.

3.2.2 Functional

3.2.2.1 Requirement 1: Add new resource collection

Introduction This service is to provide for the addition of new Resource Collections to the generic information retrieval system. This will involve the use of a Resource Access Mechanism (RAM), a generic description covering the access mechanism that deals with the resource collection storage implementation specifics. In the scope of this document we will be assuming that our resource collection is a document collection.

Input Adding a Resource Collection will entail providing a RAM to access the Data Resources within the new Resource Collection. This RAM will most likely be resource specific.

Processing The Resource Collection will be accessible by a RAM. This will give standard access to the Data Resources within and provide a level of abstraction with regards to the physical storage. The RAM is therefore responsible for a normalized view of the Resource Collection. This is the first step in the process of providing a normalized view to the Representation Domain and eventually to provide the Searcher with an answer to its query.

Output The Generic Information Retrieval System will have access, through a RAM, to the information contained in the Data Resources of the new Resource Collection.

3.2.2.2 Requirement 2: Modify resource collection scheme definition

Introduction This is a change to the description of a Resource Collection, something that can be seen as an external event for these requirements. This is a change made outside of the system being analyzed, but this will have an effect within the system. Changing a Resource Collection Scheme Definition means that the meta-data with regards to the Resource Collections content has been adjusted.

This leads to the question as to when changes to an existing Resource Collection will be propagated to the system. The change of a Resource Collections population will be covered in section 7.3. We will consider the change to a Resource Collection Scheme Definition as being separate from the population of the change, but this requirement

will most likely initiate a chain of events from requirement 2 to requirement 3 to reflect the changes.

This service provides for the changing of an existing Resource Collection Scheme Definition. This will allow for addition of new extensions to the way information within an Resource Collection can be organized for external access. A Resource Collection Scheme gives meaning to the contents within the collection by providing a mapping of the available aspect and domain information contained in the Data Resources within the Information Resources that are found in the Resource Collection.

Input A modification has taken place in the Resource Collection Scheme Definition, a change in aspect and domain information.

Processing The modified aspect and domain information will be made available to the Normalization process, so that the generic information retrieval system can be kept up to date with respect to available Information Resources. This update mechanism will be maintained within the Representation Domain, which means that the Normalization process will be responsible for updating changing Resource Collection Scheme Definitions. The frequency will depend on the time needed to update the population of the Resource Collection to reflect this change in the Resource Collection Scheme Definition.

Output The new aspect and domain information reflecting change to a Resource Collection Scheme Definition has been made available to the Normalization process. The Representation Domain may or may not reflect this change, depending on the frequency chosen to update such a change. This output will lead to a chaining effect to the following requirement described in section 7.3.

3.2.2.3 Requirement 3: Modify resource collection population

Introduction This service will allow for a change in the contents of a Resource Collection, that is the Information Resources within an existing Resource Collection. To remain flexible, this could follow the change of a Resource Collection Scheme Definition, see section 7.3. It is also possible that this can proceed a modification of a Resource Collection Scheme Definition and therefore require a change in a Resource Collection Scheme Definition. In the later case, this requirement will be followed by the application of Modify Resource Collection Scheme Definition.

Input New Data Resource(s) are added to existing Information Resource(s) within a Resource Collection. Alternately, new Information Resource(s) are added to a Resource Collection.

Processing This change in content will need to be made available to the Normalization process, this can be achieved by chaining the requirement Modify Resource Collection Scheme Definition to a change in content. Therefore the content change will be followed by an update to the aspect and domain information (Resource Collection Scheme Definition) and then made available to the Normalization process. It will remain the Normalization processes task to update its own information with regards to available aspect, content and domain information.

Output The new Data Resource(s) and Information Resource(s) in the modified Resource Collection are available to the Searcher.

3.2.2.4 Requirement 4: Add new aspect domain definition

Introduction This service is to provide for the creation of a new Aspect Domain Definition, one which will give the initial view of an Information Resource to the system. This definition will describe how the Data Resources within an Information Resource present their content (i.e. physical structure) to the system, what the domain is (i.e. postscript, text, XML) and the aspect to be made available (i.e. full-content, abstract, keywords).

Input At least a minimum (enough to make an Information Resource available to the system) Aspect Domain Definition will be supplied. Furthermore, the Information Resource must be in the system for the new Aspect Domain Definition to work with and there exists no Aspect Domain Definition for this Information Resource.

Processing The provided Aspect Domain Definition will describe the content, domain and aspect to be provided by the Data Resources within the new Information Resource. It is the responsibility of the Aspect Domain Definition to provide a Normalization for this new Information Resource.

Output A normalized view of the new Information Resource(s).

3.2.2.5 Requirement 5: Add translation definition

Introduction This service is to provide for the addition of new Translation Definitions from one Aspect Domain to another with the results that a new Aspect Domain becomes available within the system. This will lead to new Data Resources from existing Data Resources.

Input There exists an Aspect Domain Definition that provides a representation of a Data Resource within an Information Resource. A new Translation Definition is provided that maps from this existing Aspect Domain to a new Aspect Domain.

Processing The existing Aspect Domain provides the basis for a mapping from one Aspect Domain to another. The Translation Definition will provide the mapping function, which will also ensure that the newly mapped Aspect Domain is made available next to the existing Aspect Domains.

Output There exists a new mapping from one Aspect Domain to a new Aspect Domain, resulting in new Data Resources for the Searcher.

3.2.3 Non-Functional requirements

3.2.3.1 Non-Functional 1: Heterogeneity

The generic information retrieval software architecture should not be limited to any specific platform. The initial setup will be as a Free Software environment to promote the continued development and usage by as wide an audience as possible. Therefore the generic information retrieval system will make use of existing standards with regards to networking, communication protocols, storage, query languages, object orientation and existing Free Software.

3.2.3.2 Non-Functional 2: Evolvability

The generic information retrieval software architecture should be responsive to the future development of new information retrieval technologies and research needs. It should be open to new needs, new services and new facilities that will require an environment to provide inter-operability between these new technologies. This is a more important aspect than efficiency and quality of service.

To achieve this will require that the components used have to be very self containing with well defined interfaces to their environment. The most logical method to ensure this is to think in Objects and Services to be provided within the software architecture. This will lead to the separation of functionality and allow for the most flexibility.

3.2.3.3 Non-Functional 3: Inter-operability

The generic information retrieval software architecture should support the ability to interconnect and communicate with various components in various implementations. The various implementations of information resources (databases, collections, objects) and the components that will communicate with them will need to support a wide variety of inter-operability. This will be achieved with the use of distributed computing concepts such as middle-ware, standard Internet protocols, standard query languages, standard networking interfaces and well defined interfaces between component layers.

For example, the Resource Access Mechanism (RAM) will be left out of the scope of this architecture to provide for inter-operability. By stating that the RAM must provide communication over a network using standard protocols and query languages, we force this part of the

architecture to maximize its inter-operability. The components from within the architecture will only need to maintain communication via the standards mentioned to inter-operate with the various RAM's.

3.2.3.4 Non-Functional 4: Extensibility

The generic information retrieval software architecture will need to provide a stable core architecture to support the basic integration and inter-operation services needed by information retrieval system developers and researchers. On top of this, it will need to be extensible for future development, usage and research projects that may vary in the direction they take within generic information retrieval research. Currently this architecture is only directed at making information available (supply), but eventually in the future it will have to take into account the request for information (demand).

The entire architecture will be set up to facilitate easy system development for AIO research prototypes. There will be a need for well defined interfaces between components from the different layers of the Generic Information Retrieval System to facilitate the addition of new components. This can be achieved by applying available standards such as CORBA, ORB and Java to name but a few. Once again the goal is to remain flexible but still achieve the goal of prototyping for generic information retrieval research.

3.2.3.5 Non-Functional 5: Performance

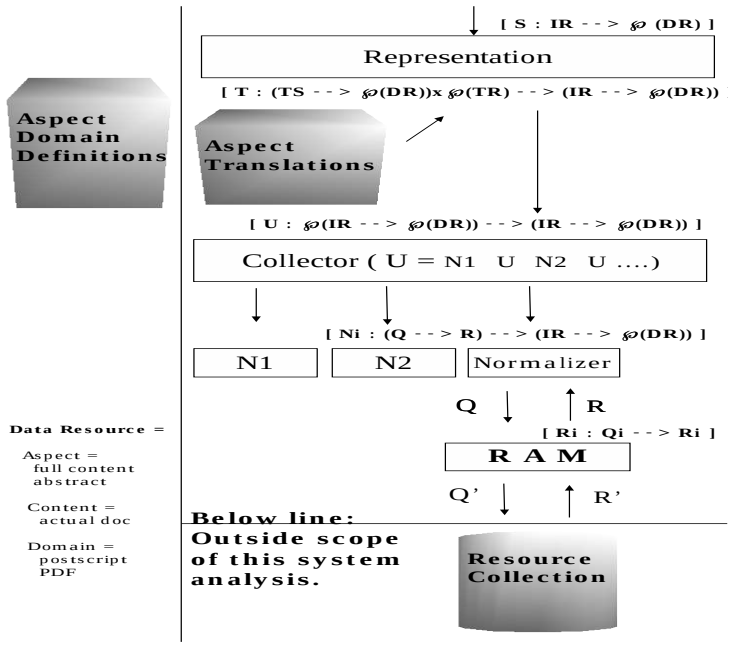
Due to the possibility of very large amounts of data in a resource, the computational power needed to process this will become a performance issue. The generic information retrieval software architecture should provide for a basis with which to conduct the desired generic information retrieval research projects and provide continued performance as needed in the future. The architecture will try to use existing building blocks for the storage and retrieval of data to provide satisfactory (and proven) performance.

3.2.3.6 Non-Functional 6: Security

The information being transported could conceivably contain sensitive data that needs to be protected. These issues will be dealt with using existing open security standards to provide the best security available over time. This is in line with the open standards mentioned in other non-functional requirements.

3.3 Architectural reference model

This section presents the generic information retrieval reference model that resulted from the requirements analysis. This is the same component view that is shown in section 3.1 with the additional details on the information supply:



Refer to appendix A in section 7.1 for detailed definitions of the listed components.

The goal in this reference model is shown in a view of the available resources as a function, $S : IR \rightarrow \mathcal{P}(DR)$ with its construction being described below from the resource collections up to the representation layer. Furthermore, there is but one normalizer in the figure that shows paths down to a RAM and eventual resource collection. This is to keep clutter in the diagram to a minimum, but in reality there will be many normalizers working with many resource collections. I have depicted several normalizers here with N1 and N2 to represent the existence of more paths to resource collections. The arrows, RAM's and resource collections have all been left out of the diagram for clarity.

Query navigation, described here from the bottom up, shows the build up of a reply set to a search request. It begins with the resource collections where the data is collected. The data is normalized, the results sets are collected, maybe transformed and finally they are made available to the searcher.

The process begins at the bottom with the results of a query into a resource collection, which is formulated as:

$$R_i : Q_i \rightarrow R_i$$

For a given information resource $i \in IR$, the results R_i are returned from the query Q_i. This will be the results after the query has been passed through the Resource Access Mechanism (RAM) and gathered the required information from the information resource. The results will be passed up to the Normalizer which has the task of col-

lecting the various query results from a particular resource collection and providing a normalized view of the results:

$$N_{i} : (Q \to R) \to (IR \to \mathcal{P}(DR))$$

For a given information resource $i \in IR$, the query results $(Q \to R)$ are normalized by N_{i} to provide a view of the information resources which is the power set of the data resources $(IR \to \mathcal{P}(DR))$ within a single resource collection. The following step is to pass these results to the collector, which then collects all the relevant normalized results into one union $U = N1 \bigcup N2 \bigcup ... \bigcup N_{i}$. The power set of the results of all queries collected from the normalizers:

$$U : \mathcal{P}(IR \to \mathcal{P}(DR)) \to (IR \to \mathcal{P}(DR))$$

These unified results are then passed up to the searcher after possibly being transformed to meet the aspect specifications of the searcher. For example, a searcher will request a set of documents that conform to a particular search criteria and then want to have the results conform to the aspect 'abstract' while the search results are in the form 'full-content.' A translation function will then be applied:

$$T : (TS \to \mathcal{P}(DR)) \times \mathcal{P}(TR) \to (IR \to \mathcal{P}(DR))$$

This will provide the translation of all result data resources from the aspect 'full-content' to 'abstract' before they are passed up to the searcher as:

$$S : IR \to \mathcal{P}(DR)$$

This is the final result of the query formulated by the searcher S, that delivers a power set of data resources conform the search criteria from a queried information resource $IR \to \mathcal{P}(DR)$.

To look at the process from the searcher on down to the resource collection requires only that the process described above be reversed. This completes the process of a query being submitted by a searcher and its migration through the generic information retrieval system at an information supply level.

Chapter 4

Reference architecture

In this chapter I will further expand on the architectural reference model discussed in section 3.3 to develop a reference architecture that maps "...software components and the data flows between the components." [Bass98].

To develop a reference architecture will involve the creation of a feature diagram and the application of three use cases. I start by giving a brief introduction to domain analysis, discuss feature modeling and then describe the feature diagram that I obtained from my analysis. I will then apply the feature diagram to three use cases to show the various components, responsibilities and data flows through the generic information retrieval architecture. This will leave us with the resource access reference architecture for generic information retrieval systems that will be validated in the next chapter.

4.1 Domain analysis

The definition of domain analysis states that it is "...the process of identifying, collecting, organizing, and representing the relevant information in a domain, based upon the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within a domain."[SEI02]. While I have mentioned that there were no existing systems to examine, there were plenty of experts on hand and an existing requirements document that gave enough form to be able to create use cases that were representative of possible existing systems. Armed with these, it was possible to analyze the resource access domain and create a feature model, which will be discussed in the following section.

The basic use of domain analysis is as an "...activity for building reusable (software) components."[SEI02]. Through the usage of domain models, feature models and architectures, it is an attempt to build reusable systems. Here the emphasis is on reuse of components and on the concept of 'product lines'.

A product line is a way of looking at the systems that can be developed by searching for requirements that are common in the systems to

be created within a line of products. This is taking a look at system development as a factory process when discussing the creation of software 'products'. For example, one tries not to look for the requirements for a single software system, but for a family of software products. This facilitates the usage of generic components that can be plugged into each product as necessary, meaning more reuse and less software development over the long run.

4.1.1 Feature modeling

Feature modeling is covered in detail by [Czarnecki00]. I choose to use this aspect of a domain analysis as it "...provides us with an abstract (because it is implementation independent), concise, and explicit representation of the variability present in the software."[Czarnecki00].

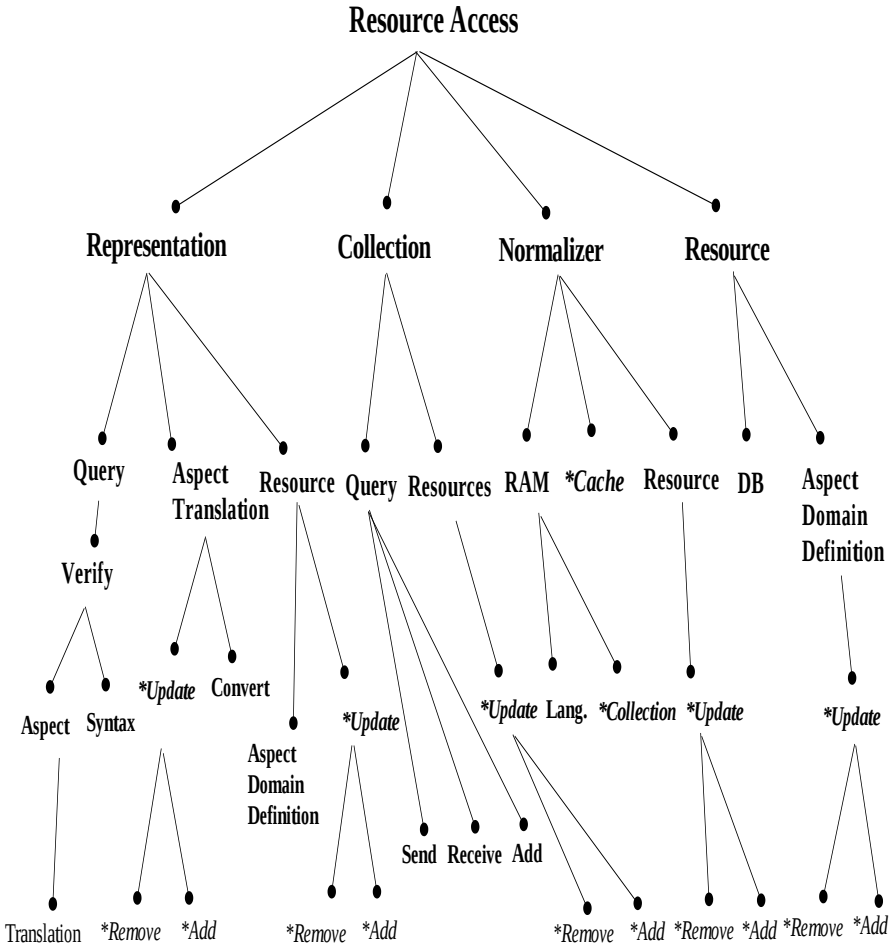
A feature model represents the common and variable features of concept instances and the dependencies between the variable features. This is further specified when stated that a feature model represents the intention of a concept, that is what a feature should do in the software system.

I will use a modified version of the feature model called a feature diagram. The diagram is in a tree form with the edges notating the various options that are available to each feature. The top level denotes the entire architecture and the second level shows the four components that have been identified during the analysis phase. My feature diagram consists of a set of nodes and a set of directed edges. The root node represents the concept of resource accessing. The remaining nodes represent features, either mandatory or optional , which will be discussed in detail in the following section. Features that are mandatory have edges leading to them representing the AND relationship as noted in the literature. The optional features have OR edges leading to them[1]. See the mentioned literature for a more complete description of the notations that can be used when drawing feature diagrams.

4.1.2 Feature diagram

Starting with the root node that denotes the resource access architecture you will find the entire feature diagram, with 'mandatory' features in normal text and 'optional' features designated with **italics*:

[1]Note that no special distinction is made in the diagram between AND or OR edges, this will be made clear where needed in the following discussions.



*Optional features.

Feature Diagram

4.1.3 Features

I will be discussing the various features from left to right in the feature diagram. Starting at the top level features (Representation, Collection, Normalizer and Resource), the discussion will then deal with each sub-feature until the leaf nodes are reached.

4.1.3.1 Representation

This feature is mandatory and consists of several sub-features, all of which are required. This feature is responsible for checking the query as it enters the system, both syntax and if the aspect requested can be returned. Also it's checking if the aspect that is not available can be produced through translation(s) and keeping this information up to

date when resources are modified. The query, aspect translation and resource features are mandatory for this feature.

Query

The query is listed as a mandatory feature under the Representation feature as it is useless to think of a resource access architecture without a query, as it would have nothing to do! Therefore I have chosen to model it as a feature, but there will be no mapping to a component in the architecture other than that the query follows the path laid out in the use case maps through the system. It has one mandatory sub-feature, verify. This feature has two sub-features to check the syntax and aspect information. If the aspect requested is not available, then the sub-feature translation provides a look-up mechanism.

Aspect translation

This feature supplies the ability to look up translations from one aspect to another, which is included as a mandatory sub-feature called convert. This sub-feature would then take care of the translation(s) needed to convert one aspect to another. Finally, the optional sub-feature update is offered to allow for dynamically being able to add or remove aspect translations to the system. It would be nice to be able to detect the new or removed aspect translations as new aspects become available in new or removed resource collections.

Resource

This mandatory feature represents the need to be able to watch the resources of the system. The mandatory sub-feature aspect domain definition (ADD) represents the keeping of a list of available aspects based on the offered resources. The optional sub-feature update is the dynamic registering of changes to the offered resources in the system. This functionality is provided by the optional sub-features remove and add.

4.1.3.2 Collection

A mandatory collection feature supplies the ability to take an incoming query, determine whether a query can or needs to be split into smaller parts, determine which normalizers are needed to access the resources for a (sub)query and pass the split queries to the next layer in the system.

Query

The query feature is also mandatory and relates closely to the one in the representation feature in that it represents the physical query movement. Here there is the need for several sub-features, all mandatory, such as receiving and being able to decide which query to send to what

normalizer for further processing. Also the sub-feature add is here to represent the collecting of the replies and being able to put the result sets for the original query back together.

Resources

This feature represents the mandatory need to be able to keep track of the available normalizers in the system. The optional sub-feature update is to supply this normalizer information dynamically when a change occurs in the resource collections. This functionality is supplied through the optional sub-features remove and add.

4.1.3.3 Normalizer

The mandatory normalizer feature is required to provide a normalization of the query and RAM access before passing the query on for further processing. The sub-features are the mandatory RAM and resource, along with an optional caching feature.

RAM

Providing for the selection of the correct RAM to access the chosen resource is the main function of this feature. It also provides, through the sub-feature language, for the translation to the language of the resource to be queried. The optional sub-feature collection reflects the possibility of dynamically updating the a list of available RAM's.

Cache

An optional feature that will be discussed in greater detail later, this feature is an important performance enhancement. Here the divided query results are cached for reuse, which will provide a better response time to later queries.

Resource

The mandatory resource feature represents the list of available resources and RAM's that can be updated by optional sub-features update, add and remove.

4.1.3.4 Resource

The mandatory resource feature should not be confused with the resource collections themselves, as these are outside the scope of the architecture. This feature refers to the ability to interact with the various other features to supply information about the availability of the physical resources. Dynamically updating the changing availability of resources, aspect information, normalizers, RAM's and translations are the main functions that this feature will supply.

DB

This feature represents the physical resource collections list, which will be the basis of any information that would need to be passed to other features.

Aspect domain definition

Here the various domain definitions are kept in-line with the conditions for the various resource collections, such as aspect information, normalizer information, RAM language information, etc. This can be dynamically updated by the optional sub-features update, remove and add.

4.2 Use case maps

The use cases are based on a generic query that enters our system at the representation level, proceeds downwards to end at the resource collection where the query will be filled with a results set. The results start then from the resource collection, enter the random access mechanism (RAM) and proceed upwards to finish with a reply to the users query. This describes the general path taken through the three use case maps, but each use case map will be outlined in detail with component descriptions and responsibilities in the following sections. This discussion covers the reference architectural level with only general components and responsibilities being discussed.

4.2.1 The maps

The use case map is a tool to link both behavior and structure in a visual way, attempting to bring understanding to complex applications. To make the reading experience a little easier I will provide a small explanation of the elements I use in my maps. For a more complete overview of the use case maps syntax, the reader is referred to Appendix A of the UCM Quick Reference Guide[Amyot99]. Use case maps were chosen as the architectural description language to concentrate on the high level view, as it is "...a scenario-based technique for capturing behavioral and to a lesser extent structural aspects of an architecture"[deBruin01].

The first element is the *path*, a black labeled line through the use case which I use to denote the path of both the query and reply as it passed through a use case. There are round end points that are the *starting points*, where the system is entered, and flat ends are the *end points* of a path. Rectangles are used to show *components*, these will most often be labeled with a name. Within a component there can be an ***X***, which denotes a *responsibility*. These are actions that can take place on the query or reply as it passes through, if needed the responsibilities will be further explained in the following discussions.

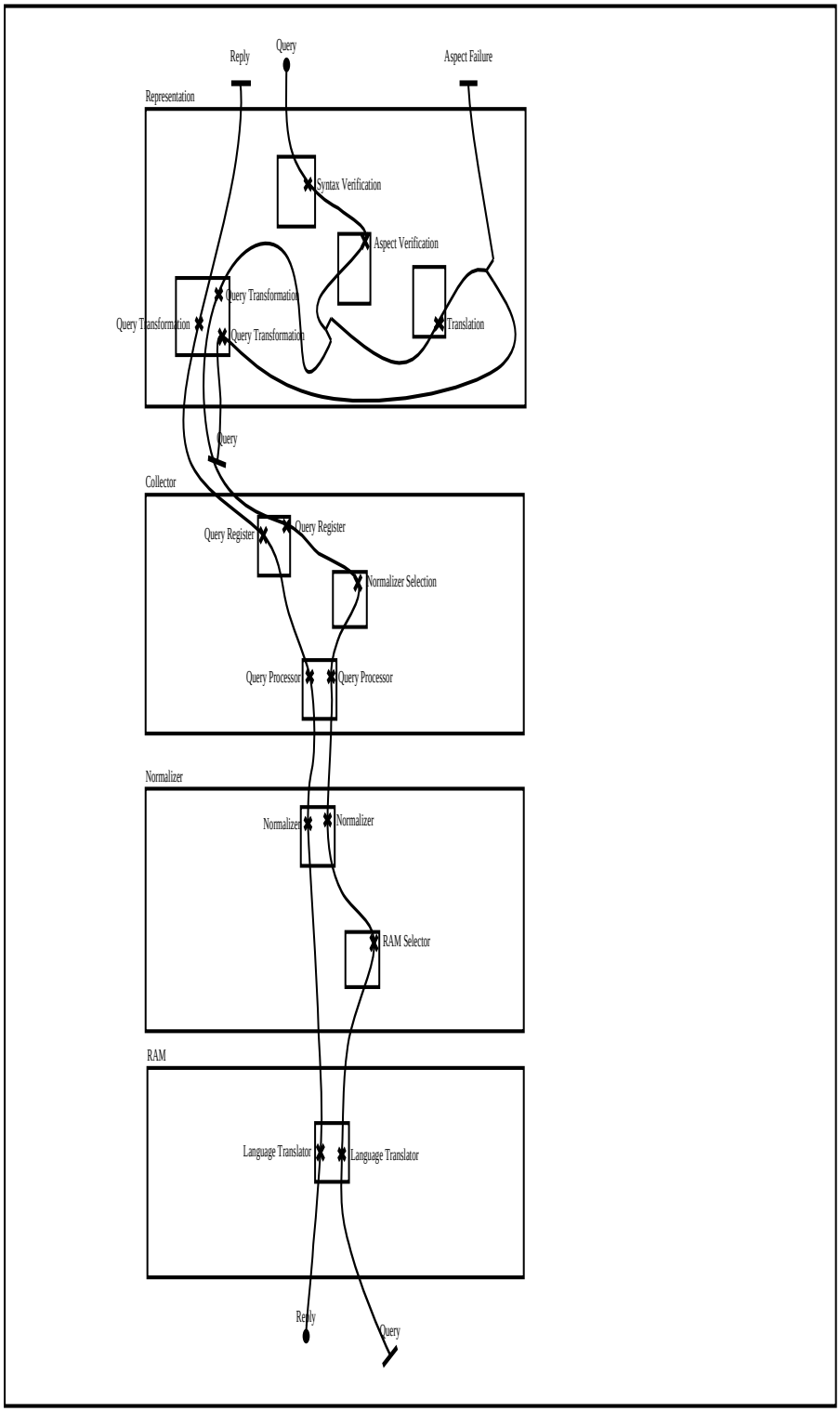
I will begin with a single resource search which is a simple static environment. The second use case map will display a resource search in an environment in which the resources are dynamically changing; new resources being added, existing ones being modified and resources being removed. Finally, the last use case map will show the buffering of queries and their results. This will show the increasing complexity that can be encountered within our generic resource access architecture and provide the foundations for a generic resource access reference architecture.

Each use case is presented with an overview figure showing the components, responsibilities and data flow of the query and reply through the resource access architecture. This provides us with the defined mapping of components and data flow that is our reference architecture. Later these three use case maps will be refined to provide a single reference architecture that will then be used to map software components in the form of stubs and plug-ins, providing a good level of generality to the resource access software architecture.

4.2.2 Single resource

This use case covers resource access with only one Resource Collection consisting of a finite set of three resources. There is no need to worry about searching for the right translation, normalizer or RAM as these are static. The rest of this section will outline the two paths in the use case, the query path and the reply path. The user query path will be followed from its entry point into the use case and following it to the end point as it exits the RAM. The reply path will then be traced back from its entry point in the RAM to exiting back to the user. The single resource access is displayed in the following use case map (see next page):

Single resource



4.2.2.1 Query path

Representation component

The users query arriving at the Representation component is the begin point of this path. The query enters the component and is syntax validated. After this the query is checked for Aspect compatibility, that is, that the system can provide the reply in the Aspect form requested (full content, keywords, abstract...). If the Aspect can be provided then the query proceeds on to Query Transformation. If not then the query proceeds to Translation, which is responsible to check if there exists a translation to the requested Aspect. If this is not possible a message is sent to notify the user that the query cannot be processed and it ends here. If it is possible to provide a reply with the requested Aspect, then the query is passed to the Query Transformation. It is the responsibility of the Query Transformation to disperse queries to the Collector and note the possible aspect translations for the replies. The query then passes to the Collector component.

Collector component

The collector component controls the splitting and delegation of the query to different Normalizers based on a check with the Normalization list. It starts with the Query Register noting the composition of the query before being split for processing. Then the query passes through Normalizer Selection which provides a list of normalizers to select from. It is the responsibility of the Normalizer Selection to provide normalizers that can provide replies to the query (in part or in full). This list is obtained via the Aspect Domain Definitions database and in this use case contains aspects for but one resource collection. Finally the query passes to Query Processor for splitting into the various parts that have been determined to provide the various replies needed to build a reply to the query. These are registered here and then passed out of the various Normalizer components for further processing.

Normalizer component

The query is potentially passed to several different Normalizer components for processing, but here we will discuss the processing of a single Normalizer. If more would be needed, they handle the query analog to the following description. The Normalizer is responsible for picking up the incoming query and normalizing it for a specific RAM. The RAM is selected to facilitate communication with a specific resource and the query is then passed on to the selected RAM.

RAM component

The single function of the RAM is to translate the query (or part of the query that it has received) into a form that can be handled by the resource that it will pass the query to. The query is translated in the

Language Translator to a query language and passed out of this architecture and on to the resource itself.

4.2.2.2 Reply path

RAM component

The single function of the RAM is to translate the reply in the Language Translator from the resource language back into a form that can be passed up to the Normalizer for further processing.

Normalizer component

The reply is passed straight into the Normalizer which has the responsibility to normalize the reply before passing it up to the Collector component.

Collector component

The reply is received by the Query Processor which notes the return of a sent query and passes the results on to the Query Register. It is the responsibility of the Query Register to put the various replies back together into a single reply set that fulfills the original query. Once this has been done the reply is passed back up to the Representation component.

Representation component

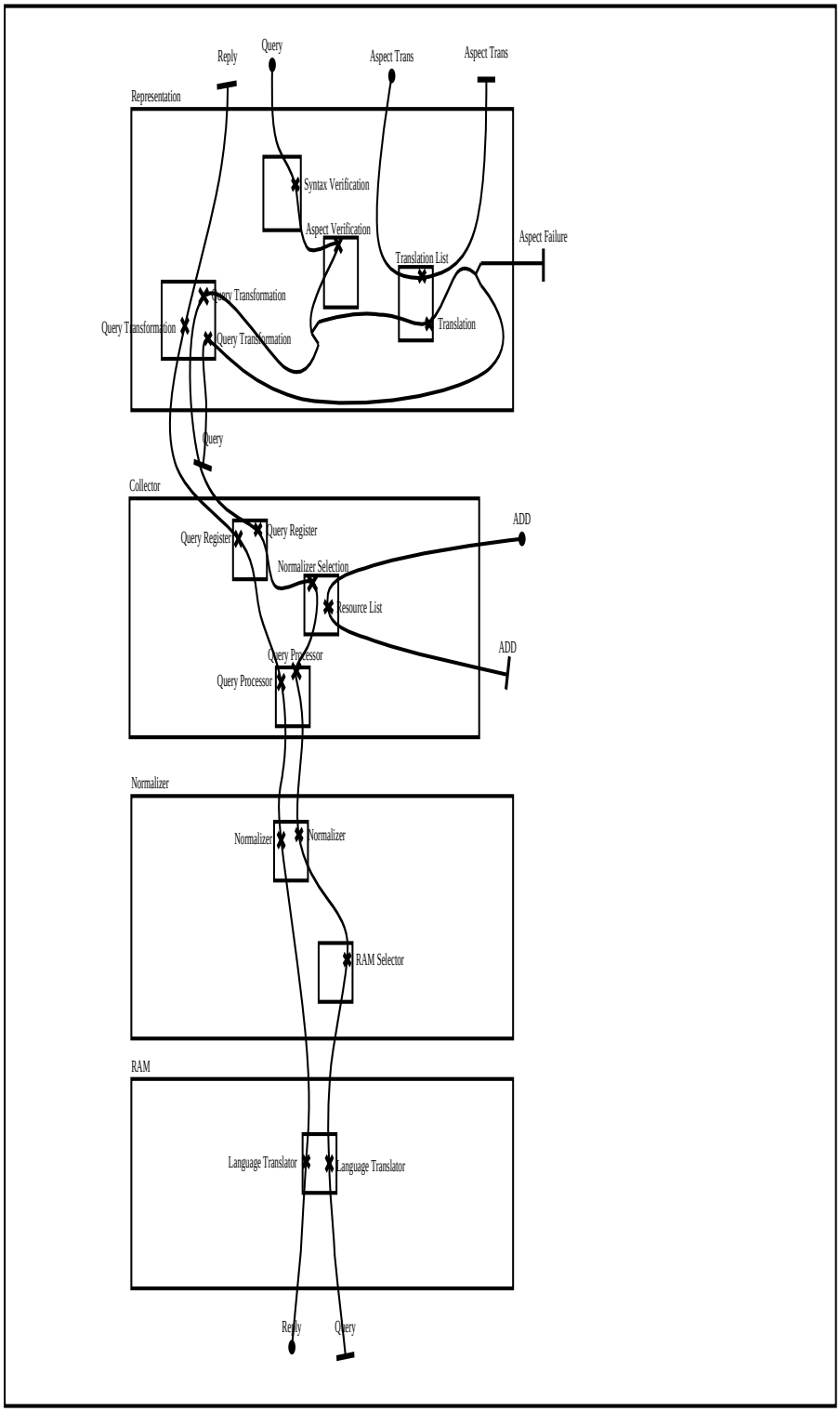
The only responsibility left for this component is to note the arrival of the reply and check if there was an aspect translation needed to satisfy the users query. If not then the reply is passed back up to the user. If it does need to be manipulated, then the aspect translation is applied and the results are passed on to the user.

4.2.3 Dynamic resources

This use case covers resource access with a dynamic set of Resource Collections. This requires that the aspect translations be updated as resource collections are updated. This also requires that the list of available normalizers also be updated dynamically as resource collections are updated. These are both accomplished by checking the Aspect Domain Definitions.

The rest of this section will outline the paths in the use case; the query path, reply path, aspect transition path and the aspect domain definition (ADD) path. The dynamic resource access is displayed in the following use case map (see next page):

Dynamic resources



4.2.3.1 Query path

Representation component

The users query arriving at the Representation component is the begin point of this path. The query enters the component and is syntax validated. After this the query is checked for Aspect compatibility, that is, that the system can provide the reply in the Aspect form requested (full content, keywords, abstract...). If the Aspect can be provided then the query proceeds on to Query Transformation. If not then the query proceeds to Translation, which is responsible to check if there exists a translation to the requested Aspect. If this is not possible a message is sent to notify the user that the query cannot be processed and it ends here. If it is possible to provide a reply with the requested Aspect, then the query is passed to the Query Transformation. It is the responsibility of the Query Transformation to disperse queries to the Collector and note the possible aspect translations for the replies. The query then passes to the Collector component.

Collector component

The collector component controls the splitting and delegation of the query to different Normalizers based on a check with the Normalization list. It starts with the Query Register noting the composition of the query before being split for processing. Then the query passes through Normalizer Selection which provides a list of normalizers to select from. It is the responsibility of the Normalizer Selection to provide normalizers that can provide replies to the query (in part or in full). This list is obtained via the Aspect Domain Definitions database and in this use case contains aspects for but one resource collection. Finally the query passes to Query Processor for splitting into the various parts that have been determined to provide the various replies needed to build a reply to the query. These are registered here and then passed out of the various normalizer components for further processing.

Normalizer component

The query is potentially passed to several different Normalizer components for processing. If more would be needed, they handle the query analog to the following description. The Normalizer is responsible for picking up the incoming query and normalize the query for a specific RAM. The RAM is selected to facilitate communication with a specific resource and the query is then passed on to the selected RAM.

RAM component

The single function of the RAM is to translate the query (or part of the query that it has received) into a form that can be handled by the resource that it will pass the query to. The query is translated in the Language Translator to a query language and passed out of this architecture and on to the resource itself.

4.2.3.2 Reply path

RAM component

The single function of the RAM is to translate the reply in the Language Translator from the resource language back into a form that can be passed up to the Normalizer for further processing.

Normalizer component

The reply is passed straight into the Normalizer which has the responsibility to normalize the reply before passing it up to the Collector component.

Collector component

The reply is received by the Query Processor which notes the return of a sent query and passes the results on to the Query Register. It is the responsibility of the Query Register to put the various replies back together into a single reply set that fulfills the original query. Once this has been done the reply is passed back up to the Representation component.

Representation component

The only responsibility left for this component is to note the arrival of the reply and check if there was an aspect translation needed to satisfy the users query. If not then the reply is passed back up to the user. If it does need to be manipulated, then the aspect translation is applied and the results are passed on to the user.

4.2.3.3 Aspect translation path

This path applies to the representation component and is taken dynamically when there is a change in the aspect information in the aspect domain definition database. Within this component the aspect translation updates the Translation List, which contains the possible aspect translations provided by the Aspect Translation DB. This updated information has no time constraints, therefore at this level of abstraction we consider the change to have taken place as soon as the aspect information is available.

4.2.3.4 Aspect domain definition path

This path applies to the collector component and is taken dynamically when there is a change of resources in the Aspect Domain Definition (ADD) database. Within this component the ADD updates the Resource List, which contains a list of dynamically changing Resources, which includes the Normalizers for Resource access. This updated information has no time constraints, therefore at this level of abstraction we

consider the change to have taken place as soon as the resource information is available.

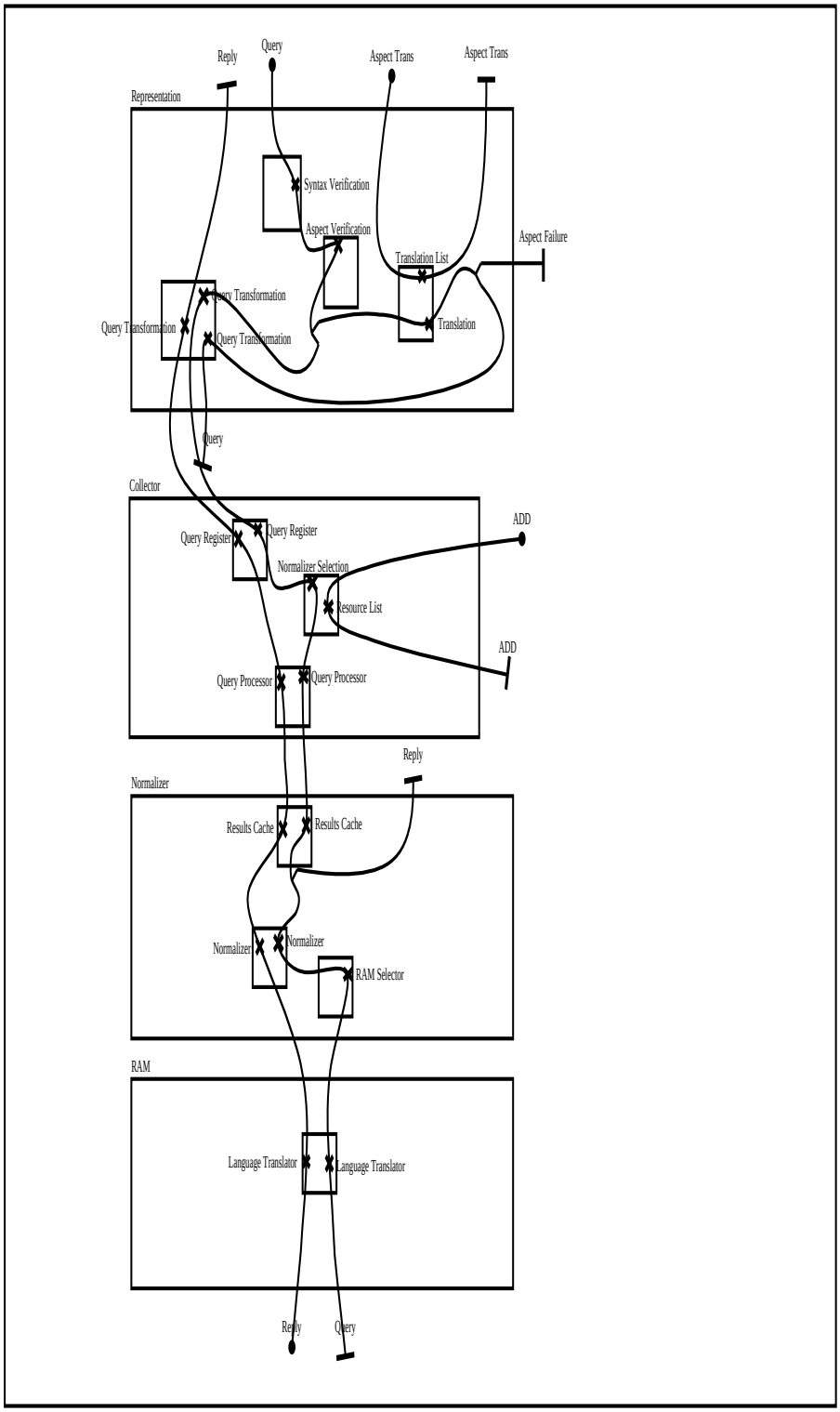
4.2.4 Buffered dynamic resources

This use case covers resource access with a dynamic set of Resource Collections. This requires that the aspect translations are updated as resource collections are updated. This also requires that the list of available normalizers also are updated dynamically as resource collections are updated. These are both accomplished by checking the Aspect Domain Definitions.

The rest of this section will outline the paths in the use case; the query path, reply path, aspect transition path and the aspect domain definition (ADD) path. The user query path will be followed from its entry point into the use case and following it to the end point as it exits the RAM. The reply path will then be traced back from its entry point in the RAM to exiting back to the user. The aspect transition and aspect domain definition paths will be covered for the two components where they appear in the use case.

The single most important difference between the previous use case maps is the addition made to the Normalizer component. This will be explained in more detail in sections 4.2.4.1 and 4.2.4.2. The dynamic resource access is displayed in the following use case map (see next page):

Buffered dynamic resources



4.2.4.1 Query path

Representation component

The users query arriving at the Representation component is the begin point of this path. The query enters the component and is syntax validated. After this the query is checked for Aspect compatibility, that is, that the system can provide the reply in the Aspect form requested (full content, keywords, abstract...). If the Aspect can be provided then the query proceeds on to Query Transformation. If not then the query proceeds to Translation, which is responsible to check if there exists a translation to the requested Aspect. If this is not possible a message is sent to notify the user that the query cannot be processed and it ends here. If it is possible to provide a reply with the requested Aspect, then the query is passed to the Query Transformation. It is the responsibility of the Query Transformation to disperse queries to the Collector and note the possible aspect translations for the replies. The query then passes to the Collector component.

Collector component

The collector component controls the splitting and delegation of the query to different Normalizers based on a check with the Normalization list. It starts with the Query Register noting the composition of the query before being split for processing. Then the query passes through Normalizer Selection which provides a list of normalizers to select from. It is the responsibility of the Normalizer Selection to provide normalizers that can provide replies to the query (in part or in full). This list is obtained via the Aspect Domain Definitions database and in this use case contains aspects for but one resource collection. Finally the query passes to Query Processor for splitting into the various parts that have been determined to provide the various replies needed to build a reply to the query. These are registered here and then passed out of the various normalizer components for further processing.

Normalizer component

The query is potentially passed to several different Normalizer components for processing. If more would be needed, they handle the query analog to the following description.

The Results Cache has the responsibility to check if the query is in the cache. If not the Normalizer will process the query for a specific RAM. The RAM is selected to facilitate communication with a specific resource and the query is then passed on to the selected RAM. If the results are in the cache, the reply is immediately sent back to the Collector component and the Normalizer component is finished.

The reasoning behind buffering the results only at the Normalizer component level will be covered at the end of this section.

RAM component

The single function of the RAM is to translate the query (or part of the query that it has received) into a form that can be handled by the resource that it will pass the query to. The query is translated in the Language Translator to a query language and passed out of this architecture and on to the resource itself.

4.2.4.2 Reply path

RAM component

The single function of the RAM is to translate the reply in the Language Translator from the resource language back into a form that can be passed up to the Normalizer for further processing.

Normalizer component

The reply is passed straight into the Normalizer which has the responsibility to normalize the reply before passing it up to the Results Cache. This responsibility will cache the results and then pass the reply up to the Collector component.

The reasoning behind buffering the results only at the Normalizer component level will be covered at the end of this section.

Collector component

The reply is received by the Query Processor which notes the return of a sent query and passes the results on to the Query Register. It is the responsibility of the Query Register to put the various replies back together into a single reply set that fulfills the original query. Once this has been done the reply is passed back up to the Representation component.

Representation component

The only responsibility left for this component is to note the arrival of the reply and check if there was an aspect translation needed to satisfy the users query. If not then the reply is passed back up to the user. If it does need to be manipulated, then the aspect translation is applied and the results are passed on to the user.

4.2.4.3 Buffering in the normalizer component

At this point in the search the query has been processed and split for specific normalizers, if we cache here we can do this per resource collection. For example, imagine a query has a resource access system with six different resource collections numbered from one to six. A user sends query Q1, which is processed and split to sub-query resource collections 1, 4 and 5. These result sets are then cached in the Normalizer component. The next user query Q2 arrives, which processes

out to sub-query resource collections 2 and 5. The first sub-query (2) will have to be passed down to the resource collection but the other (5) will be taken from cache and passed up immediately.

Should this buffering be done at a higher level then a finer grain of resource usage would be prevented. In our above example you would then need to wait for a user query to request information that is exactly 1, 4 and 5 as a set. This is not as likely to happen as a sub-query needing just one of the sub-query results. Buffering at a lower level is done within most database solutions to some degree to optimize query times, therefore that is also ruled out.

Furthermore, the reply has been normalized before caching, which should lead to better response times and higher chance of reuse than if the results were cached at a higher level.

4.3 Reference architecture

After looking at the three use cases in the above sections, we can now complete our architectural journey with a reference architecture for resource access within a generic information retrieval system. The final use case was the cumulation of the various features listed in the feature diagram in section 4.2.4 and this will be the reference architecture examined in the following chapter.

Chapter 5

Proof of concept

In this chapter the validation takes place through application of the reference architecture from chapter 4 to the resource access side of the generic information retrieval system discussed in chapter 2. This is done in the form of a proof of concept, with the mapping of features to software components depicted within a single use case map.

The mapping from the reference architecture to a software component architecture can be seen as a process used to reach a solution to the mandatory (and optional) features based on the requirements that have been discussed previously. In the following sections a buffered dynamic resource collection is used as the final software architecture. This discussion will present the various components, sub-components, stubs, plug-ins and responsibilities within these plug-ins.

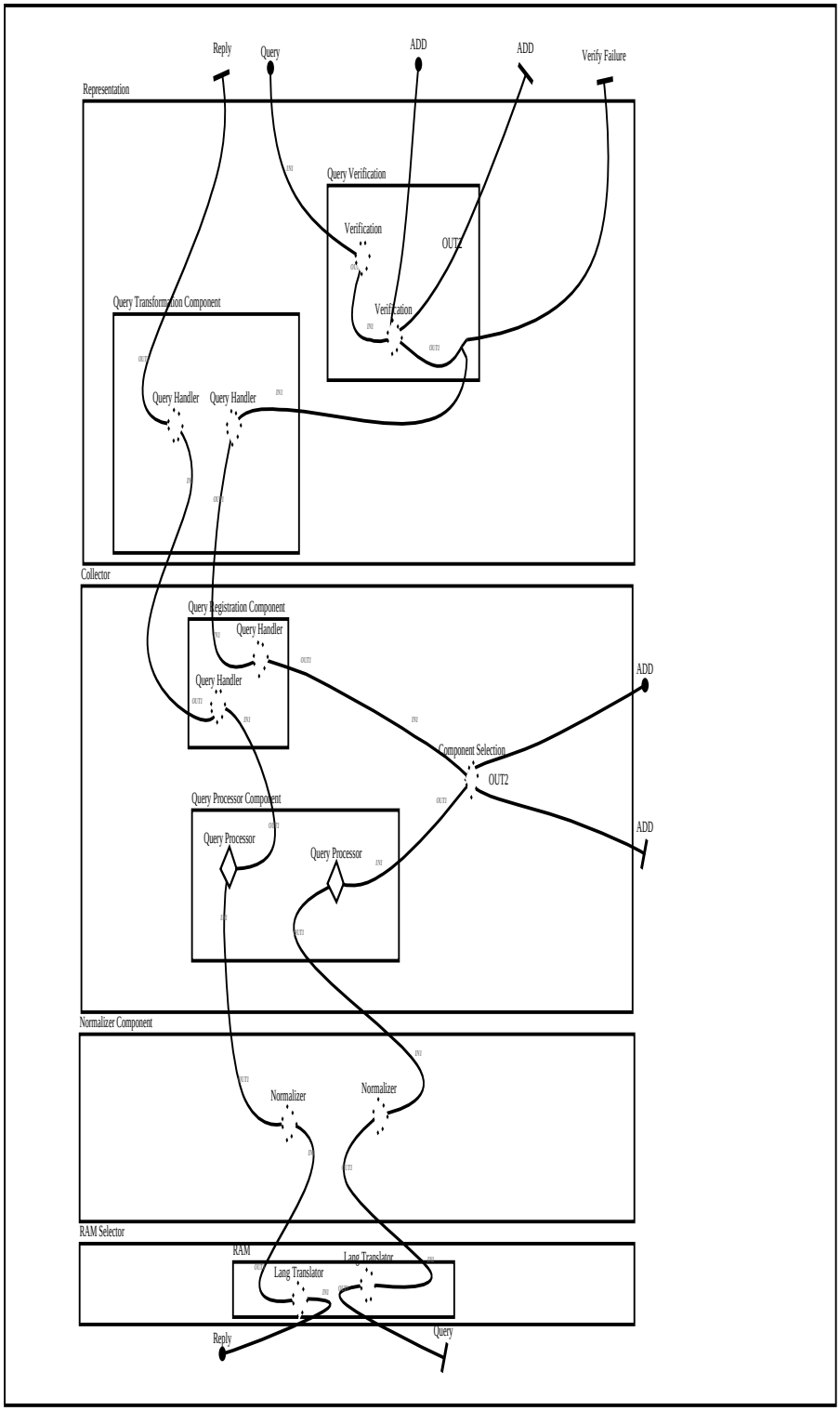
The chapter will end with a software architectural proposal for use in resource access within generic information retrieval systems.

5.1 Mapping the architecture

The features discussed in the previous chapter will be mapped to the reference architecture. Within the map there has been one notational feature added called a *stub*, which I will explain briefly here.

When mapping becomes complex it is often desirable to provide another layer of mapping which simplifies the top level map by displaying a stub. This stub is then worked out in its own map called a *plug-in*, using the same notations as describe previously such as components, paths and responsibilities. These stubs can be represented with a solid diamond to represent a *static stub*, or with a diamond consisting of dashed lines called a *dynamic stub*. The static stub represents but one plug-in for that stub. The dynamic stub can be filled with one or more plug-ins based on certain system configurations. I will attempt to state clearly in the following discussion why a static or dynamic plug-in is used and what some of the possible plug-ins could be. The figure presented here is the reference architecture from section 4.3, with the software plug-ins identified with stubs. First, an overview of the software architecture (see next page):

Software component overview



The various stubs shown in the previous figure will have plug-ins available and a feature overview will be given of the four main components in the following section. For reference a table is supplied with the software components, sub-components and stubs to be discussed in the rest of this chapter:

Base components		**Sub-components**		**Stubs**
Representation	->	Query verification	|	verification
	->	Query transformation	|	query handler
Collector	->	Query registration	|	query handler
			|	comp. selection
	->	Query processor	|	query processor
Normalizer			|	normalizer
RAM selector			|	lang. translator

5.2 Features to components

The components discussed here will be organized in a top down format as a query enters the architecture. I begin with the representation component and discuss the stubs and related plug-ins, then the collector component and its stubs with its plug-ins, etc.

Each base component discussion will begin with a table showing the overview of the features that will be discussed. The link between the feature diagram and the various implementing plug-in components, starting with the mandatory and proceeding to include the optional features, will be covered as they pass the review. To clarify the notation used, **bold** is for base features, **italics* are for optional features and lower cased features are the leaf nodes in that branch of the feature diagram. Following the feature discussion I will present the components, stubs and available plug-ins.

5.2.1 Representation component

This component has the responsibility to accept queries from the user space, verify these queries and take care of any aspect translations that might need to be done. The returning reply is then checked for possible aspect translations, translated if necessary and presented back to the user.

Features

Taking a look at the representation component one gets the following feature overview:

Base feature		**Sub-features**						
Representation	->	Query	->	Verify	->	Aspect	->	trans.
					->	syntax		
	->	Aspect Trans.	->	**Update*	->	**remove*		
					->	**add*		
			->	convert				
	->	Resource	->	ADD				
			->	**Update*	->	**remove*		
					->	**add*		

The rest of the discussion will take us through the representation component, detailing the sub-components, stubs, plug-ins, responsibilities and feature mapping.

Mapping

This component is made up of two sub-components called Query Verification and Query Transformation Component. These map the aspect transformation and resource features. The query feature is listed to show the need for the representation component to handle a query, therefore you will only see the query in the architecture as a path.

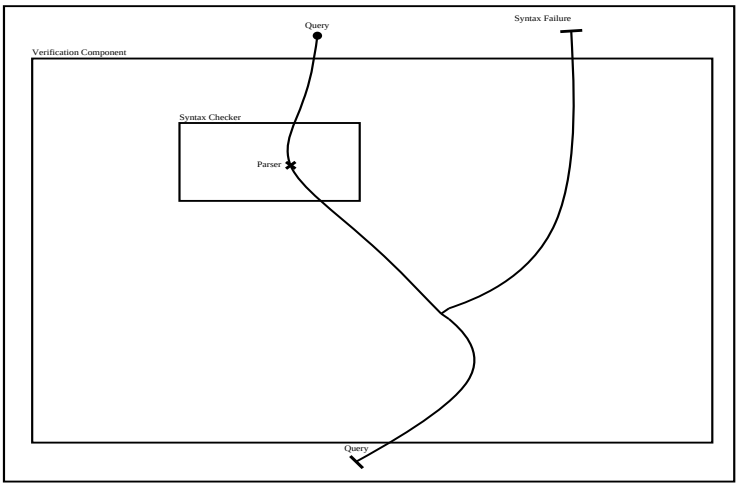
Verification component

The query verification component has the responsibility to verify the syntax and aspect. These are shown as dynamic stubs, which means there is more than one option when choosing the plug-ins for these components. The stub is listed twice for clarity in the map, but they represent a choice of one of the following plug-ins:

Verification plug-ins		**Short description**
Syntax verification	-	checks query syntax
Aspect verification static	-	checks aspect in static db
Aspect verification dynamic	-	checks aspect in dynamic db

To check the syntax of a query it would be easily possible to have more than one plug-in based on one per query format that might be allowed. This supports future extensibility of the system, but for now I have included a basic plug-in as follows:

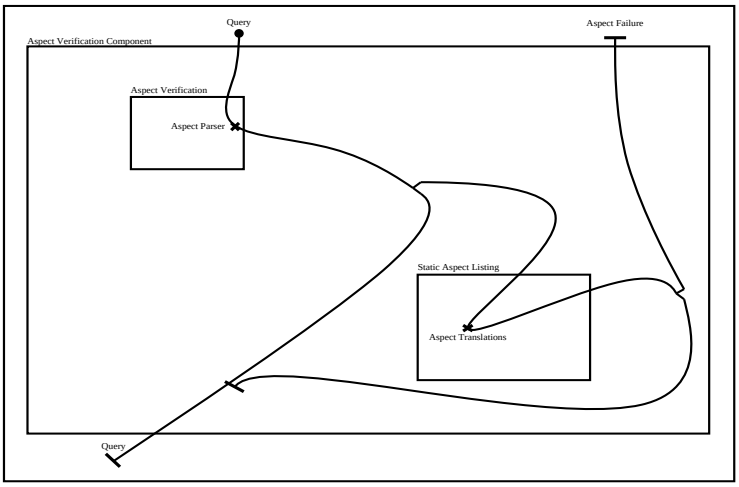
Plug-in: Syntax verification component



This plug-in consists of a syntax checker that uses a parser with the responsibility of parsing the query. If successful it will be passed on to the next phase in the system and if it fails a message goes back to the user. Again, for any query format that the system might need you could develop a plug-in to check that specific syntax. This provides for future system evolution due to the self-contained nature that these plug-ins represent. They may be replaced with a minimum of adjustment to the system.

After checking the syntax the query then arrives at the aspect verification stub which has one of two options, static or dynamic aspect verification:

Plug-in: Aspect verification static

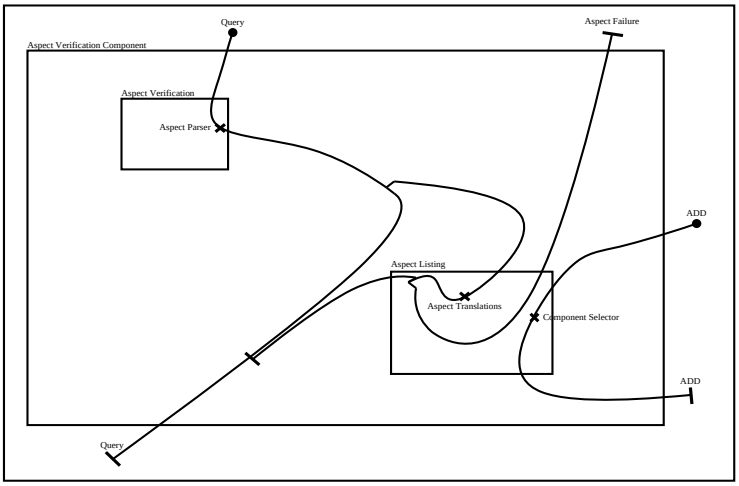


When dealing with a static collection of aspects, such as a small

system or one meant for specific purposes, the aspect verification static plug-in would do the job nicely. The query is checked against the readily available aspects, if this matches then the query is passed on without further actions. If the aspect is not readily available, but reachable through one or more translations from the static listing of aspects, then the query is listed for translation upon returning with a reply set. Here we see the mapping of the convert feature to the representation component architecture.

This information will be passed on to the query handler, to be covered next. Should the aspect be unavailable, either through direct listing or a translation, then a failure message would be sent to the user stating the problem. The aspect translations are considered here to be static in that they are not updated. There is a listing in the aspect domain definition database, but this is a static list. This feature is integrated in the aspect translation responsibility.

Plug-in: Aspect verification dynamic



When dealing with the possibility of being able to dynamically update the aspect translations then one needs a bit more flexibility. This is provided for in the aspect verification dynamic plug-in. The added component selection responsibility will keep track of the aspect translations that change with the modification of the aspect domain definition database. When a resource collection is added, part of the definition will include the provided aspect(s) and translation(s) that come with it. This is only necessary when taking advantage of the optional update feature in the representation component.

This two stage verification should provide a reasonable amount of security to the system by preventing overloading of the networks with bogus queries or bogus aspect requests. By checking this in the first stage, most likely close to the users physical location, it can be prevented that any extra network load be generated by bad queries being introduced into the system.

Also by adding the optional feature of updating the aspect translations we see an indirect mapping of the resource feature. The aspect domain definitions feature and optional updating feature are visible in the dynamic version of the plug-in, but the resource feature is more virtual and exists behind the scenes.

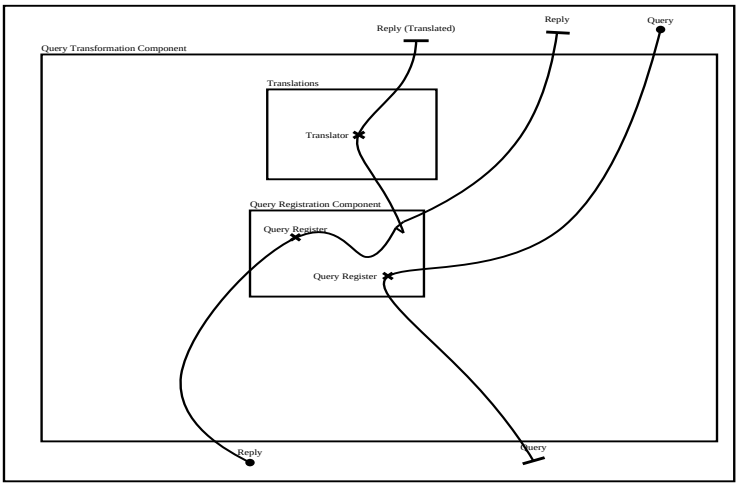
Query transformation component

Finally, arriving at the Query Transformation component we see the dynamic stub Query Handler which has the following plug-in:

Transformation plug-in		Short description
Query transformation	-	registers query, trans. aspect as needed

Once again, to keep the system extensible there have been no restrictions placed on the number of plug-ins that can be defined for this stub. I have supplied an example that shows the simple responsibilities as follows:

Plug-in: Query transformation



In this plug-in the query is registered by the query register, which notes if an aspect translation is needed once the reply set returns. If so, then the reply will be passed back up to the translator responsibility which will take care of the aspect translations before passing the results onward. If the translation is not needed, then the query will bypass the translator and be passed straight out of this plug-in.

The convert feature is actually performed here in the translator responsibility, completing our tour of the feature mapping for the representation component.

5.2.2 Collector component

The tasks of registering incoming queries, selecting the appropriate normalizers, splitting of the query into sub-queries if necessary and reassembling the reply sets are handled by the collector component.

Features

The complete listing of collection component features is as follows:

Base feature		**Sub-features**				
Collection	–>	Query	–>	send		
			–>	receive		
			–>	add		
	–>	Resources	–>	**Update*	–>	**remove*
					–>	**add*

The rest of the discussion will take us through the collection component, detailing the sub-components, stubs, plug-ins, responsibilities and feature mapping.

Mapping

This component is made up of two sub-components called Query Registration Component and Query Processor Component. Furthermore, there is a separate stub that is labeled Component Selection. The query registration takes care of noting the incoming queries before they are split and passed on. The component selection is used to select the correct normalizers for the eventual (sub-)queries that are to be passed on down the architecture. The query handler then takes all this information and splits the query into smaller parts if needed and sends them out to the selected normalizers for further processing.

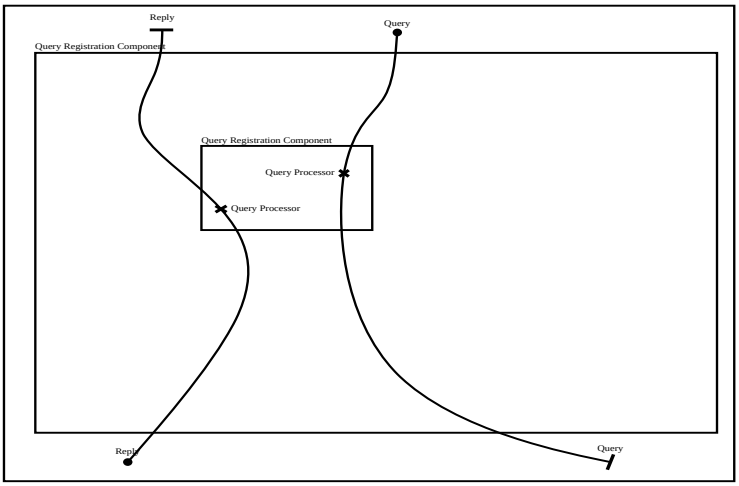
Query registration component

The query registration has the responsibility to register the query before it gets split up into sub-queries. This supports the features send, receive and add. Sending is to pass the query onward, receive is to collect the returning replies and add will then ensure that the original query is put together before heading back up into the system. The stub query handler is shown as two stubs to provide a clear representation of the query and reply paths. The following plug-in has been defined, but by keeping the stub dynamic I have allowed for extensibility of the system to include future usages:

Query handler plug-in		**Short description**
Query handler	-	registers query before split occurs
	-	assemble replys

The plug-in is defined as follows:

Plug-in: Query handler



It consists of one component that registers the query, this is taken care of by the query processor responsibility and ensures that the reply can be pieced back together to provide the user with a complete reply set. Here we see the query send, receive and add features mapped into the component architecture. Some of these features will also be encountered in the query processor component below.

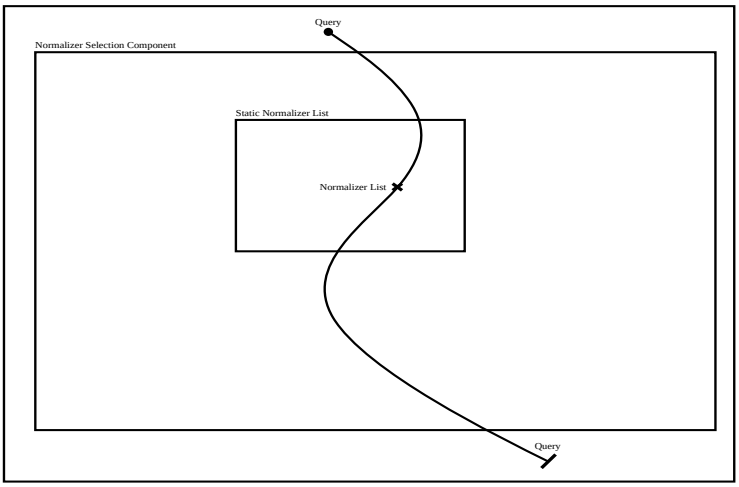
Component selection

The component selection is a dynamic stub that can be filled with one of the following plug-ins based on the needs of the system:

Component plug-ins		**Short description**
Normalizer selection static	-	selects from static list
Normalizer selection dynamic	-	selects from dynamic list

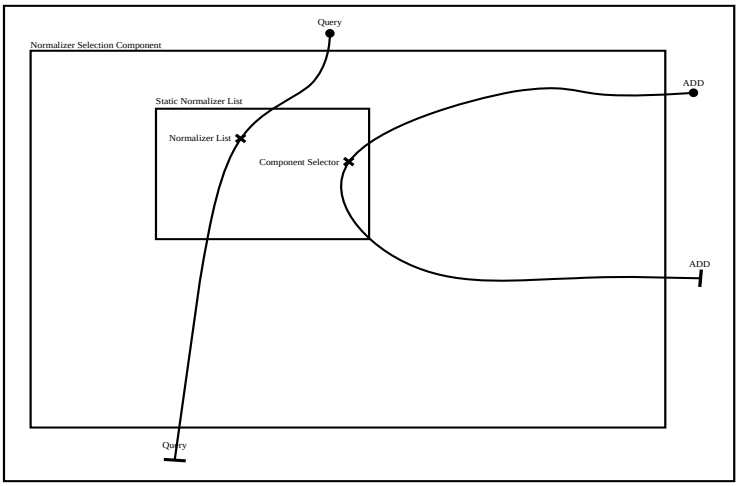
Here I have listed but two possibilities as far as the normalizer selection plug-ins go, one for simple systems and one that contains a bit more complexity. The stub is dynamic to facilitate both extensibility and evolvability within the architecture.

Plug-in: Normalizer selection static



This plug-in makes a normalizer selection from a static list that is not updated. This selection is based on the query registration component suppling information as to what resources access the query will need.

Plug-in: Normalizer selection dynamic



The dynamically updated normalizer list will change the process a bit, showing the indirect mapping of the resource feature. The updating is done dynamically via the updated information in the aspect domain definitions which details changes to the resource collection. This is analog to the already discussed representation component, except that only the normalizer information will be extracted for the list here.

This is an example of inter-operability that is contained in this system, the ability to extract only the needed information that is supplied

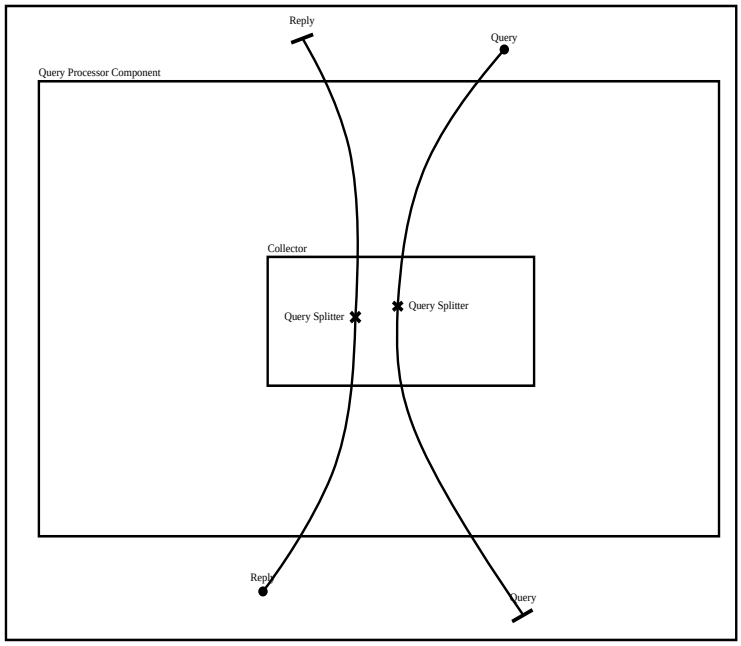
in as general a manner as possible. The aspect domain definitions are good examples of this, as we have seen in both the representation and collector components.

Query processor component

Finally the query reaches the query processor component, to be processed before passing on to the normalizer component. Once again two stubs are present, both denoting the same plug-in:

Query processor plug-in		Short description
Query prosessor	-	based on avail. normalizers, splits
	-	assembles replys for query handler

Plug-in: Query processor



A query processor consists of one responsibility, that of the query splitter. It has the responsibility, based on the information received from the previous plug-ins, to split and pass the queries on to the appropriate normalizers. Upon the return of the replies, it has the responsibility to process the queries back up to the query handler for re-assembly.

This is again part of the query feature, namely the send and receive features. The mapping of collection features is complete with the return of the reply through this plug-in.

5.2.3 Normalizer component

To normalize the incoming queries, the normalizer component will register them and select the correct RAM for access to the assigned resource. This can be done every time, or a buffered solution can be offered based on caching of the queries. Therefore the query, if found in the cache here, would be returned immediately. If not, it would be processed further in the system and the results would be cached for future references.

As one normalizer is for a finite set of resources, the queries coming in are already split up by the collector component. This means that the caching is occurring on a finer grained basis than if it were done at a higher level.

Features

The complete listing of normalizer component features is as follows:

Base feature		**Sub-features**				
Normalizer	–>	RAM	–>	language		
			–>	**collection*		
	–>	**cache*				
	–>	Resource	–>	**Update*	–>	**remove*
					–>	**add*

The rest of the discussion will take us through the normalizer component, detailing the stub, plug-in, responsibilities and feature mapping.

Mapping

The normalizer component consists of but one dynamic stub to facilitate the extensibility of this system. I will show two different plug-ins for the normalizer:

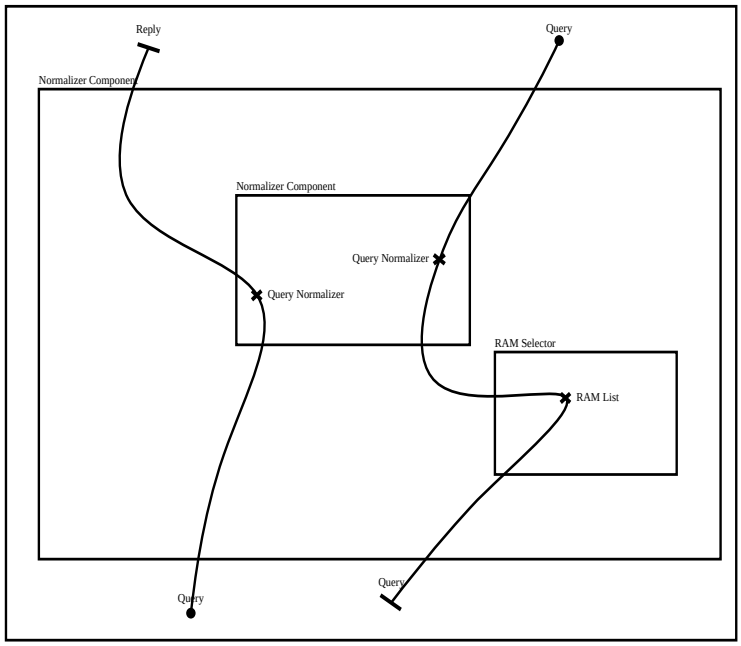
Normalizer plug-ins		**Short description**
Normalizer basic	-	registers query and selects RAM from list
	-	nomalizes reply
Normalizer caching	-	check query, if cached immediate reply
	-	register query, select RAM
	-	reply normalized and is cached

These two possibilities cover both a simple system with no frills and a more advanced system that makes use of buffering. This buffering is accomplished though the caching of the queries and their reply sets. The reasons behind this have been discussed earlier and are of interest when looking at evolvability of the system.

Normalizer stub

The single stub in this component has two possible plug-ins based on which type of system will be needed.

Plug-in: Normalizer basic

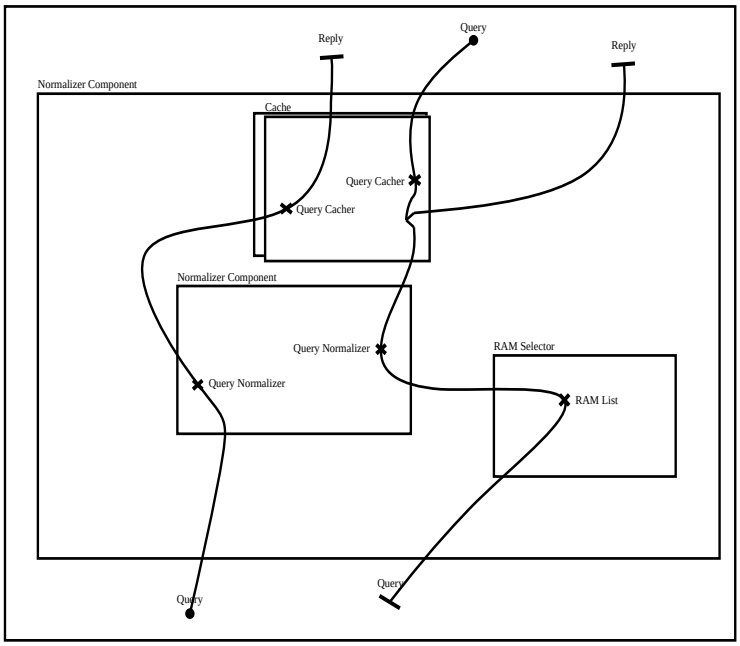


The normalizer basic component is for the low end system, one that is not too concerned with query performance. When each search can be completed in an acceptable amount of time you only have to concern yourselves with registering the passing query and selecting a RAM for the resource to be accessed. On the reply side it is only necessary to normalize the reply set before passing the results on to the collector component.

The RAM feature from our list has been mapped to the RAM list, but the mandatory language feature has yet to appear. This will be handled in the following component.

Finally, it is important to remember that using a minimalistic approach is also one of the solution options. Not all architectural solutions will have enough room in the budget or enough system resources to facilitate an extensive performance solution. This is the reason that when possible there have been simple plug-ins offered as possible solution components which should provide for more heterogeneity when the architecture can also be applied to lower end systems.

Plug-in: Normalizer caching



A higher end solution would be found in the normalizer caching plug-in, providing the same services as those listed in the normalizer simple plug-in, with one exception. The incoming query is checked for cached reply sets, if not then the query would be handled as in the normalizer simple plug-in. As the reply was returned through the normalizer caching plug-in, it would then be cached and returned up to the collector component. If the query was originally found in the buffers upon arrival in the normalizer caching plug-in, then the reply would be sent back immediately.

As mentioned before, this will be a more resource intensive solution, but some budgets and system reserves might require such performance enhancements. As the high and low end have been presented here as options, it is not hard to imagine solutions that would fit somewhere in the middle. By defining the normalizer stub as dynamic it has been made our architecture both open to evolution by allowing extensions through new plug-ins.

Finally, the required resource feature is seen here in the form of a possible extension to this dynamic stub. It would be possible to map the optional collection feature with the help of a dynamically updated list of RAM's. This could be done analog to the normalizers and aspect transitions described in the mentioned components. I have chosen not to model this in here as a third plug-in, it is left as an exercise to the reader.

5.2.4 RAM selector component

The last component provides the access needed to directly query the resource collections. The dynamic language translator stub will allow for various plug-ins that speak the resource collections native language.

Features

This component has an overlap with the features from the normalizer component, namely the RAM and language features.

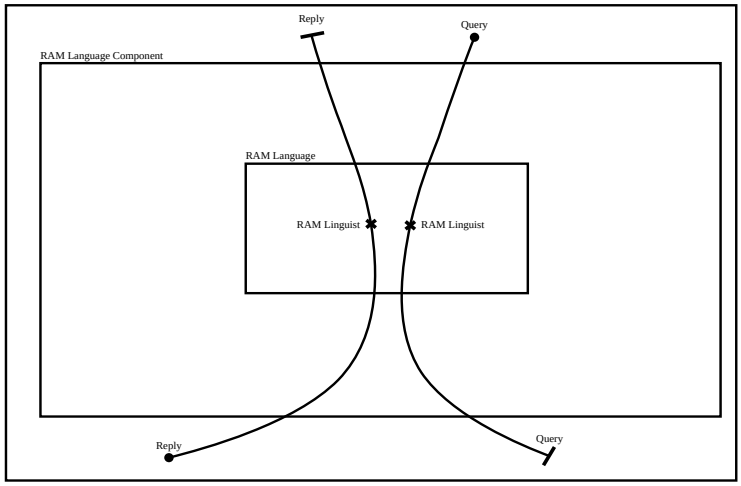
Mapping

The RAM selector component consists of only one dynamic stub, in which I have mapped one plug-in:

Language translator plug-in		Short description
Language translator	-	speaks resource collections lang.
	-	obtains the query reply set

The sole task of the plug-in is to speak to the resource collection and to return the reply in a form that the normalizer can understand.

Plug-in: Language translator



The single responsibility listed here, the RAM linguist, has the sole function to act as a translator for the normalizer component. It speaks to the resource collection in its own query language, retrieving the reply set and translating this back up to the language that the normalizer will be able to understand.

This maps the RAM and language features discussed before in the normalizer section directly to this plug-in. Furthermore, the resource

feature is handled here in such a way that, even though it falls outside the scope of this plug-in, it has direct bearing on the functions performed here. For example, the resource collection is a document database. This RAM linguist would then translate for the normalizer component so that the query would be performed as SQL and the reply set would be sent back in pseudo set theory language. As you can see, the resource feature does have direct bearing on this plug-in.

Finally, by keeping the communication protocols abstracted through these RAM linguists it is possible to allow the inter-operability of many different types of resources within this architecture. This also makes a good case for promoting heterogeneity in possible system solutions based on this architecture.

5.3 Results

The question now is, are the requirements in section 3.2 provided for in this proof of concept with the mapped features to components, stubs and plug-ins? Let us take a closer look at both the functional and non-functional requirements as applied to our proof of concept.

5.3.1 Functional requirements

The first requirement was to be able to add a new resource collection. This was covered in both the representation and collector components so that the addition of new resources will lead to a dynamic updating of the aspect translations and normalizer lists. This occurs through the updating that will be pulled in via the aspect domain definitions database.

The second and third requirements are to allow the modification of a resource collection scheme definition and population. This is the mapping of aspect information to the aspect domain definition database. I have accounted for this by allowing updates of this information to propagate dynamically into the representation component. This would also be partly covered in the collector component when we update the normalizers list, as this is part of the aspect domain definition database information. Furthermore, it was also discussed when dealing with the dynamic updating of the RAM listings in the normalizer component. This would also be updated dynamically via a change in the population, such as the removal of a certain resource. That would result in the removal of that specific RAM as it would no longer have a resource to communicate with.

The fourth and fifth requirements are to allow for the addition of new aspect domain definitions and aspect translations. These would apply to the representation and collector components that dynamically update their aspect information. For example, this was taken care of by the normalizer selection dynamic plug-in for the collector component and described in detail above.

5.3.2 Non-functional requirements

How about the non-functional requirements? These are traditionally the tougher ones to nail down, therefore I will not try to say that I have covered them completely, but present the approach I took.

First off, security. The main concern about any system that will become distributed is eventually security, of the data, of the systems involved and of the results the system will deliver. The solution here is in the nature of this system, that is of prototype development support in information retrieval research. I would submit that the security can be planned into the system by taking a sand-box view, that is, keeping the system on a scale that will be used inside of one environment. By keeping the entire system sand-boxed it provides adequate security for our purposes.

Secondly, evolvability. This system has been designed with evolution in mind, by choosing object orientation, self-contained software components and plug-ins. By not committing to any one development scheme, or any one programming language it is hoped to achieve a broad base for future growth. The model is outlined, not the implementation specifics.

Third, by not limiting the communication between components (it has been left up to the implementor as to which protocols will be used) to anything other than open specification there is a large base for inter-operability between the solution components. Whether they will be sending messages, TCP/IP, HTTP, FTP or p2p, it is all possible and therefore a highly inter-operable solution.

Heterogeneity was touched upon in the proof of concept. By keeping the various component architectures based on nothing platform dependent, there is nothing standing in the way of having each component (even each plug-in) running on a different platform. This includes the provided solutions for a wide range of systems, both low and high end solutions were offered.

Finally, extensibility. I have been trying to predict the future by defining the resource access architecture as broad as possible. I hope to have taken into account as much eventual new development in the information retrieval world as possible. This is a nice try, and only time will tell, but history has often showed us that we always forget the simple things. I hope that my application of simple component architecture whenever possible will have made future extensions as painless as possible.

5.4 Solution architecture

Here I will present a possible solution architecture that fills some of the components, plug-ins and responsibilities with possible software solutions. For easy reference, I present the same table as in section 5.1, but with the solutions mapped in:

Components		Solutions
Query verification	–>	OO software
Query transformation	–>	OO software
		Databases
		TOM
		Jini[tm] networking services
Query registration	–>	OO software
		Databases
		Jini[tm] networking services
Query processor	–>	OO software
		Databases
component selection stub	–>	OO software
		Jini[tm networking services
normalizer stub	–>	OO software
		TOM
language translator stub	–>	OO software
		TOM

Object oriented software

The component structure that I have used to model the resource access domain of generic information retrieval lends itself well to the object oriented software development. As Eliens states, “It is not about programming, it is about program organization.”[Eliens95]. This methodology gives us the ability to define concise interfaces between components, use all manner of communication channels between components, encapsulate the functionality of components and ensure portability across multiple platforms.

I would recommend not trying to mix all forms of object oriented implementations (C++/Java/Small-Talk), but choose just one for implementing the research prototypes that this system is designed for. Therefore I would choose Java to support the entire non-functional requirements range. The Java motto states it nicely, ’program once, run many’ meaning write the software on one platform and it will run anywhere.

Databases

In keeping with the Free Software requirements and the open systems paradigm, there are several hundred solutions available for databases [FreeDB02]. Depending on the complexity of the solution, some of the more well known and supported Free Software project databases are MySQL [mysql02] and Postgresql [postgresql02].

While MySQL is well know for its simplicity, when dealing with transactions it is often recommended to use Postgresql database. I think it is more a mater of taste as they are both well developed with

not only free software community support but commercial support as well.

Jini[tm] networking services

To facilitate the automatic updating of the normalizer lists, the aspect domain definitions and eventual RAM lists it is easy to imagine the usage of the Jini[tm] networked services.

The basic idea is “By using objects that move around the network, the Jini architecture makes each service, as well as the entire network of services, adaptable to changes in the network. The Jini architecture specifies a way for clients and services to find each other on the network and to work together to get a task accomplished. Service providers supply clients with portable Java technology-based objects ("Java objects") that give the client access to the service. This network interaction can use any type of networking technology such as RMI, CORBA, or SOAP, because the client only sees the Java object provided by the service and,subsequently, all network communication is confined to that Java object and the service from whence it came.”[Jini02].

This would be applied to the definition of resources and the aspect domain definitions. When a new resource is added or modified then the changes would be propagated about the network in the form of the services that would be defined. For example, an aspect translation service would keep an eye out for changes to the aspect translations that are available in the system, updating as the changes occur.

Typed object model

This is the results of a PhD thesis project and provides “...a data model and a system of mediator agents that support the widespread use of diverse data formats...”[Ockerbloom98]. It is an interesting tool that would supply a methodology for working with data formats over networked systems.

This would be a useful model to provide a mechanism for aspect translations, normalizations and even for the conversion of results sets between the RAM and normalizer components. This is not an existing implementation, just the descriptions of the model along with a typed object protocol specification (TOP), therefore it would require extra work of implementation.

Chapter 6

Conclusion

While many applications make use of information retrieval systems, most do this within specific domains. For example, search engines on the Internet are mostly applied to text based resource collections or slight variations of this domain. The IRIS group at the KUN has made it one of its research goals to search for a generic information retrieval architecture, one that can search effectively within any kind of resource domain.

In this chapter I will present a summary of my look into a resource access software architecture for generic information retrieval, covering briefly the steps taken and results obtained. The original problem statement and research goals will be held up to the light to determine whether they have been met. A few conclusions will be made, a short evaluation offered and a few suggestions made about possible future directions that could be taken to expand on my thesis work.

6.1 Research summary

The problem statement that was given in chapter 1.4 states that the IRIS group at the University of Nijmegen is looking for a software architecture to facilitate prototyping for projects within generic information retrieval. This led to the research goals:

- analyze the concepts involved
- determine the requirements for prototyping within generic information retrieval architecture
- determine a reference architecture
- attempt to present a solution architecture

The first step of researching the generic information retrieval domain led to the decision to present an overview as a search for the Holy Grail of information retrieval. Without such an overview the more specific focus of resource access within generic information retrieval would have been starting without giving it any depth.

By specifically focusing on the supply side of generic information retrieval, I was able to establish a requirements document. These requirements were filtered out of interviews conducted within the staff of the IRIS group and a brain storm session. At this point the requirements were specific to resource access within generic information retrieval, but had also led to the creation of a generic information retrieval reference model.

The following step was to map the model to a reference architecture. This was done with the help of three use cases, starting with a simple single resource search and ending with a complex resource access involving buffered dynamic resources. By starting with a domain analysis, it was possible to generate a feature diagram and map the features to a reference architecture.

The generated resource access reference architecture would then be used for a proof of concept. This would be the validation of my research, by applying a mapping from the feature diagram to software components to provide various plug-ins for the resource access software architecture. At the end of this step a solution architecture was given with solution options for the main elements that were found in the proof of concept; object oriented software components, databases, Jini[tm] networking services and the TOM model.

6.2 Conclusions

This thesis has only dealt with the resource supply side of generic information retrieval, specifically resource access within a generic information retrieval architecture. It is a shame that there was not more time allotted within the Computer Science thesis project to allow for a wider analysis of this subject. I feel as if I have only uncovered the tip of the information retrieval iceberg.

One of the most difficult aspects dealt with while working on this thesis was the concept of keeping the architecture as “generic” as possible. It was easy to discuss the generic information retrieval architecture, but when one begins to define borders it is no longer generic. This wrestling with generic and specifics was the challenge that this architecture endured and the results are a balance of both worlds. At certain points in the process choices were made to make this possible, for example, to only examine document collections as resources. This is clearly no longer generic, but necessary for the completion of the task at hand.

The presented solution architecture is based on the current state of affairs within the Free Software community as of this writing. All the components are well tested and should provide for an open solution. They are not all encompassing and it will remain a matter of taste as to the programming language, database type or conversion model that will be used to implement this architecture.

A label ’work in progress’ would best describe the state of the presented architecture, with the presented solution a snapshot along the road to the final generic information retrieval software architecture. It

should be used and seen as a starting point, not the final solution. Luckily, the IRIS group will be using this research as a stepping off point in the PRONIR project, described in section 1.3, with regards to prototyping in their generic information retrieval research. This will begin with the application of a document conversion tool as touched upon in the previous discussion as Aspect conversions. The TOM model will play an important roll in this part of the implementation.

In conclusion, the delivered resource access software architecture has provided the starting point, meeting the set forth requirements, for building the prototype environment for generic information retrieval research. This will be implemented in the near future by the author within the PRONIR project.

6.3 Evaluation and future suggestions

While this thesis has delivered a software architecture for resource access within generic information retrieval, it is not the end station on the road to discovery. The following step would be to test the architecture by implementing a prototype with the suggested tools. This would provide an environment in which further experimentation could build on the results presented here. I will be doing this in the coming years as the scientific programmer for the PRONIR project, which will give me the chance to view the results of my early work for this thesis. I am looking forward to implementing the eventual generic information retrieval architecture and watching it evolve.

Some of the following suggestions are about subjects that touched this thesis project on the edges, but due to lack of time have not been discussed. To totally ignore them would not be correct, therefore they will be mentioned here as points for future consideration with pointers to applicable research.

When looking at the extensibility of the future generic information retrieval systems you would want to look at an interesting conceptual architecture describing how to deal with large, multi-component, distributed architectures. Such a conceptual architecture for information retrieval is presented in [Papazoglou01]. Also of interest is the protocol specification for communication between client and servers in the domain of information retrieval [Z39.50].

Some interesting concepts involving query structures are presented in [Hofstede96, Wondergem00]. A related topic is that of managing meta information over large distributed networks. A CORBA[tm] Meta Object Facility [Crawley99] is a specification that would be very interesting for dealing with meta data when finally implementing the generic information retrieval architecture.

Finally, although a few suggestions are given here, they too are not complete. After all, the word 'generic' in generic information retrieval could mean anything, including resource access to objects in the future that have not yet even been considered! It's this one word that makes it so hard to locate the generic information retrieval software architecture Holy Grail.

Chapter 7

Appendixes

7.1 Appendix A - Definitions, acronyms and abbreviations

Here a list is presented with the terminology that will be encountered throughout the remainder of this thesis:

- **PRONIR** - Profile based Retrieval Of Networked Information Resources.
- **IRIS** - Information Retrieval and Information Systems.
- **GIR** - Generic Information Retrieval. Denotes an environment where it is possible to build information retrieval research prototypes from modularized components. An architectural test bed for the construction of research prototypes.
- **Resource Collection** - a collection of Information Resources such as documents, database containing video files, music collection, or anything that can be considered a collection of Information Resources.
- **Information Resource** - is one element in a Resource Collection, such as a document, a video file, or anything that can be considered a single resource object.
- **Data Resource** - a single view of an Information Resource, such as a single representation of one specific document.
- **Aspect** - is a view of a given Data Resource with regards to one particular Information Resource. This describes how the information contained in the Data Resource is presented. For example, in the case of a document collection the Aspect could be full-content, keywords, or an abstract representation of the document.
- **Data Domain** - a Data Domain is the definition containing information about the access interface and data source structure for a Data Resource.

- **Aspect Domain Definition** - describes a Data Resource structure. Consists of descriptions of the Aspects and Data Domains (with the data source structure defined in the Data Domain) available from Data Resources within an Information Resource.
- **Translation Definition** - a mapping of an existing Aspect to a new Aspect, or the mapping of an existing Data Domain to a new Data Domain.
- **Searchers** - the entire user application involving the concept of profiling and the initiator of a search request in the generic information retrieval system.
- **Resource Domain** - the collection of all Resource Collections in the system.
- **Resource Collection Scheme** - lists the content and domain information with regards to a particular Resource Collection.
- **RAM** - Resource Access Mechanism is an interface that provides normalized access to the data from a specific Information Resource, such as an Unix ODBC RAM gives access to an Unix database. Therefore the RAM has specific knowledge of the underlying access protocol.
- **Representation Domain** - a unified view on the underlying Information Resources.
- **Normalization** - the process of providing a normalized view of an Information Resource, specifically a Data Resource within the Information Resource. The results of this view will be an Aspect providing a specific representation (i.e. full-content, keywords, abstract) of the underlying information. This is a process that can be seen as a function that maps an Information Resource to an Aspect, such as f:IS –> XML. This example shows a mapping of an Information Resource of documents to provide an XML document representation.
- **Normalizer** - provides for Normalization with regards to one Resource Collection.

7.2 Appendix B - Interview questionnaire

Interview Homework: Architecture for Generic Information Retrieval

The problem

Currently the research department Information Retrieval and Information Systems Group is setting up a broad based research project called PRONIR (Profile based Retrieval Of Networked Information Resources) that will involve two AIO's. One will concentrate on profiling of users and the other will focus on the representational domains. To facilitate the AIO's with building their research prototypes it is desirable to create a software architecture that meets the AIO's needs. Currently each AIO will have to create an entire software architecture to implement his/her specific prototype, something that is expensive in both time and funding.

Goals

The following goals for this thesis are:

- Determine the requirements needed for the prototyping architecture.
- Determine a reference architecture for the KUN's prototyping needs.
- Attempt to provide a solution architecture (in the form of a Proof of Concept) for the interfacing between the representational domains and the resource domains.

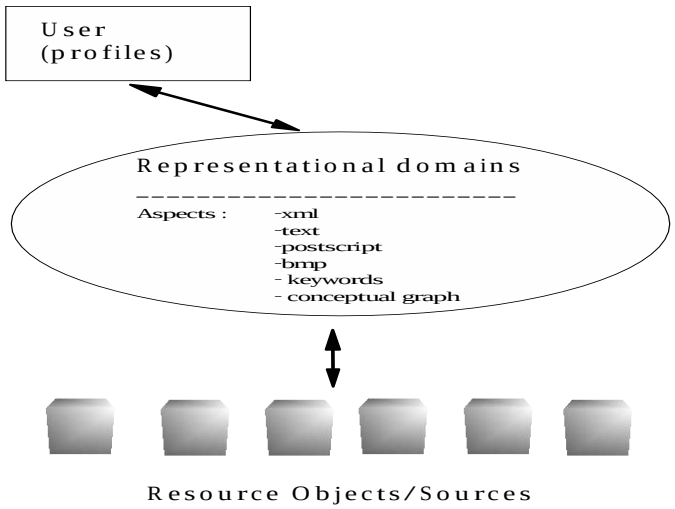
The project goals are aimed at creating a software environment, within an Information Retrieval System, that will allow prototype builders to quickly implement the components needed outside of their area of focus. This will free them up to spend more time focused on their specific research interests.

Representational Domains

The Representational Domain is visualized here in this section from the context of the Generic Information Retrieval System. The overview is presented and then discussed in a bit more detail with the focus on the Representation Domains and Resource Domains.

Conceptual layout

The high level conceptual overview of a generic information retrieval system can be seen in figure below.



Here we have the user interfacing with the retrieval system via his/her profile, which gives access to the Representational Domains. These Representational Domains can be seen as a view that is compiled for the search engine to optimize access to the underlying Resource Domains. The resource objects can be databases, remote networks, documents, video libraries, basically any sort of information that you can think of searching for over a wide network.

Representational domains

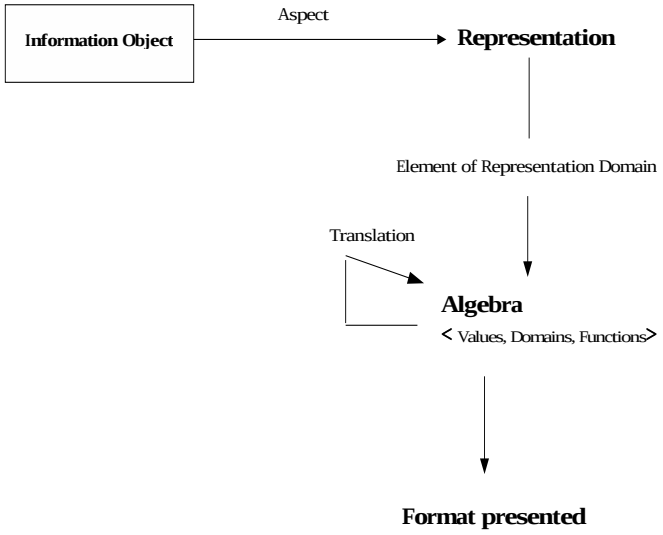
The most important fact here is that the Representational Domain is a characterization of the provided resource, which provides one or more "Aspects" to the user. Here an "Aspect" can be seen as a role that the resource plays in that form. For example, we could define a Representational Domain with an Aspect of this document as a list of keywords, as XML, or as HTML. Each "Aspect" is another role that this resource can play within our defined Representational Domain.

Resource domains

The choice of naming in Resource Domains comes from the fact that not only can a resource be a database, document, or file, but also an application that delivers some content with meaning to be represented via a Representational Domain. One should always keep in mind that the main thread in this project is to keep the architecture as 'generic' as possible, therefore a resource can be almost anything.

Adding new resource domains

One of the most basic of actions that can be performed within this conceptual model is that a new Resource Domain will be added. The process that a new Resource Domain follows can be presented as follows:



Every Resource Domain will have certain Aspects, either defined at introduction to the system or derived from other existing methods. These Aspects give us a representation of the resource that is then the an element in the Representational Domain. For example, the HTML2PS method that already exists in a system can translate an input document from HTML into a postscript document, thereby creating a new Aspect within the Representational Domain.

This process can be defined by:

- Information Sources : IS
- Representation Values : RV
- Representation Domains : RD

and if $\sum \varepsilon$ RD then $\sum$ is an algebra:

Σ = <V, R, F>

such that V $\subseteq$RV, R a set of n-arry relations over V and F is functions from V* to V.

This is but an short formal definition to start us on the path to defining Aspects, Aspect Representation Domains, Representational Domains and Translations between the two domains. This should give the reader a basic idea of how a mapping is achieved between a new Resource Domain to becoming an element in one or more Representational Domains.

Modularized system components

To facilitate a system that allows for prototyping within a Generic Information Retrieval system we require that the system components be of a modular nature. This is to allow the prototype builder (AIO) to "plug in" the parts of the software infrastructure that are not directly relevant to the area of study. For example, the research area might be to evaluate user profiling. In this case a prototype would need to implement the search engine, the resource domains and the Information Objects along with the User Profiling/Interface that is actually the focus of the project. If the components are available the prototype can be easily setup with the concentration being focused on the front end, instead of on the entire infrastructure.

Three tiered approach

There will be three levels of granularity, starting with a high level view and ending with a specific element from a Resource Domain.

The initial view will be defined at a conceptual level, with a description of mappings between the χE (Resource Domains Existential) and χI (Representational Domains Inessential).

The second view will refine this to the logical view where a mapping exists between a database with a representation of χE that via a transition is mapped to χI in another database.

The final step will then be a mapping between a document in postscript format to one in XML format, an implementation view.

Questions

This section includes a list of questions that will require your input. Please answer these questions in the space provided. The questions and answers will form the basis of the interview:

Which aspect(s) of the Information Retrieval System can be found in your research project(s)?

_____

_____

_____

_____

_____

What for Representational Domains are found in your research project(s)?

_____

_____

_____

_____

_____

What kinds/sorts of operations are used for your Representational Domains?

_____

_____

_____

_____

_____

What do the specifications look like that you use to create a Representational Domain from a Resource Domain in your research project(s) (if you can, please give short example or include a copy of documentation that describes this)?

_____

_____

_____

_____

_____

Do you think (from prior experience) that such an architecture as described here could be beneficial to you with future Information Retrieval prototypes?

_____

_____

_____

_____

_____

***** **New from Kees Leune interview** *****

1. Split the Representational Domain (figure 1) to include a level between Resource Domains and Rep. Domain called “Meta-Data”?
2. Information from the system should be presented in a generic form to the User, what should the data look like?
3. Should data presented to user be presented in various ways or only in defined set?
4. What is your interpretation of the differences between the terms “Information Retrieval” and “Information Discovery”?

7.3 Appendix C - Requirements document

Requirements for a Generic Information Retrieval Software Architecture

Eric D. Schabell

February, 24 2002

1 Introduction

1.1 Purpose

This document states the architectural requirements of a Generic Information Retrieval System for usage in the PRONIR project by the IRIS group at the KUN in Nijmegen, the Netherlands. The requirements stated serve as a basis for the acceptance procedure of this system. The document is also intended as a guide for the eventual start of the design phase.

1.2 Scope

The intended architecture will provide the software foundations to facilitate prototyping for research purposes within the Generic Information Retrieval domain. The infrastructure will provide a means for PhD student researchers to implement the complete structure needed to search for information while concentrating on their own area of research. This will reduce the time needed to implement a prototype and provide reuse of software components for future users.

The scope of this SW architectural study will be limited to the Resource section. This will include the Resource Domains on up to the Representation Domains and will focus on providing a SW architecture to facilitate the addition of new Resource Collections into the system.

1.3 Definitions, Acronyms and Abbreviations

- PRONIR - Profile based Retrieval Of Networked Information Resources.
- IRIS - Information Retrieval and Information Systems.
- IR - Information Retrieval.
- Resource Collection - a collection of Information Resources such as documents, database containing video files, music collection, or anything that can be considered a collection of Information Resources.
- Information Resource - is one element from a Resource Collection, such as a document, a video file, or anything that can be considered a single resource object.
- Data Resource - a single view of an Information Resource, such as a single representation of one specific document.
- Aspect - is a view of a given Data Resource with regards to one particular Information Resource. This describes how the information contained in the Data Resource is presented. In the case of a document collection the Aspect can be a full-content, keywords, or an abstract representation of the document.
- Data Domain - a Data Domain is the definition containing information about the access interface and data source structure of an Data Resource.
- Aspect Domain Definition - describes the a Data Resource structure. Consists of descriptions of the Aspects and a Data Domains (with the data source structure defined in the Data Domain) available from Data Resources within an Information Resource.
- Translation Definition - a mapping of an existing Aspect a new Aspect, or a mapping of an existing Data Domain to a new Data Domain.
- Searcher - the entire user application involving the concept of profiling and the initiator of a search request in the Generic Information Retrieval System.
- Resource Domain - the collection of all Resource Collections in the system.
- Resource Collection Scheme - lists the content and domain information with regards to a particular Resource Collection.

- RAM - Resource Access Mechanism is an interface that provides normalized access to the data from a specific Information Resource, such as a JDBC RAM gives access to a database . Therefore the RAM will need to have specific knowledge of the underlying access protocol.
- Representation Domain - a unified view on the underlying Information Resources.
- Normalization - the process of providing a normalized view of an Information Resource, specifically a Data Resource within the Information Resource. The results of this view will be an Aspect providing a specific representation (i.e. full-content, keywords, abstract) of the underlying information. This is a process that can be seen as a function that maps an Information Resource to an Aspect, such as f(a: IS -> XML). This example shows a mapping of an Information Retrieval of documents to provide an XML document representation.

1.4 References

- DOC1: *Profile Based Retrieval of Networked Information Resources - project proposal.*
- DOC2: *Generic Information Retrieval SW Architectural Vision.*
- DOC3: P. van Bommel, H.A. Proper, *Towards a General Theory for the Supply of Information.*

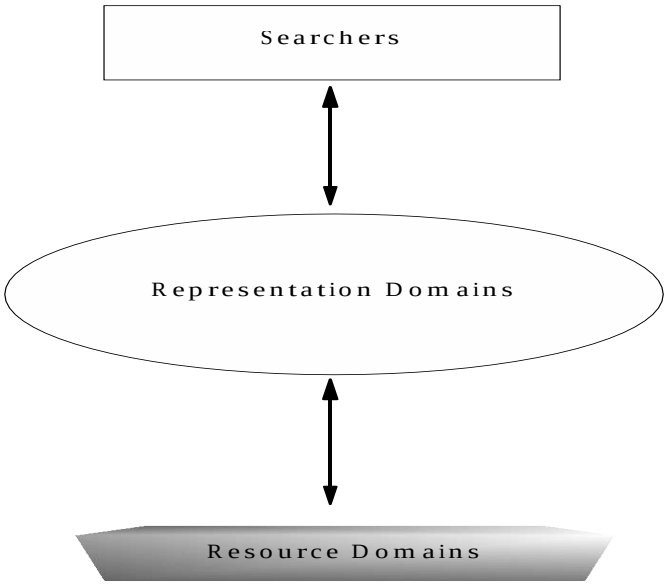
1.5 Overview

Section 2 of this document gives a general overview of the Generic Information Retrieval System. Section 3 gives the architectural requirements of the services to be offered by this system.

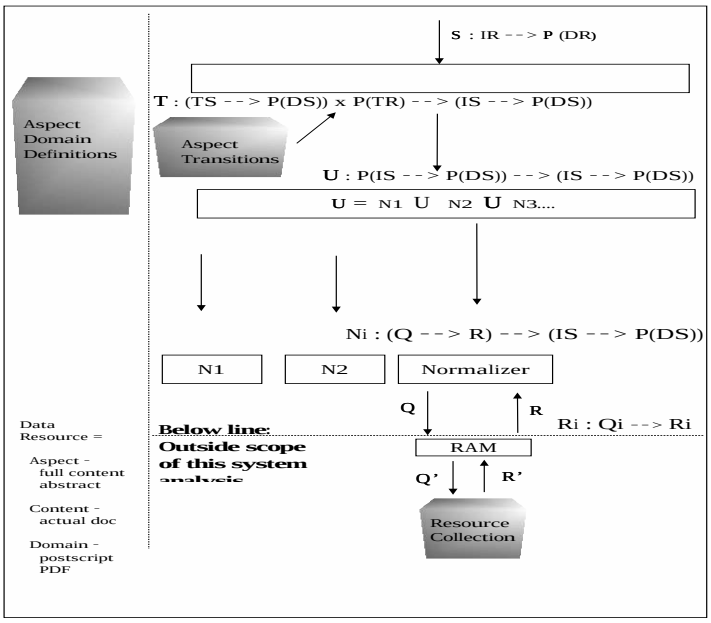
2 Overall description

2.1 Product Perspective

There is no existing product as such, but a wish to setup a structured environment for the purpose of prototyping within Generic Information Retrieval research projects, starting with the PRONIR project. The first and foremost approach is to take the 'Holy Grail' view, that is, to attempt to describe the perfect Generic Information Retrieval System. Due to the size of such a system (and the constraints on this thesis), the domain will be narrowed to focus on a subset of the whole. First a look at the Holy Grail from a conceptual level:



This view will be narrowed down to the specific level of Representation Domains and Resource Domains, as can be viewed at a more detailed conceptual level:



Within the Resource Domains there are concepts such as Information Resource, Data Resources and RAM that can be found in the definitions listed in 7.3. Within the Representation Domain we will deal with such concepts as Normalization, Aspect Translations and Aspects as described in 7.3.

2.2 Product Services

The software architecture will be looked at with respect to the services that it will provide (or not provide) to the users, developers and researchers. The services to be provided can be split as follows:

- Add New Resource Collection.
- Modify Resource Collection Scheme Definition.
- Modify Resource Collection Population.
- Add New Aspect Domain Definition.
- Add Translation Definition.

2.3 User Characteristics

The users of the Generic Information Retrieval SW Architecture will be of two classes; one being the developers whom implement Information Retrieval research prototypes and the other being the PhD. research students whom will not necessarily have extensive knowledge of working within a SW architecture. The scientific programmer should be charged with maintaining future Generic Information Retrieval SW Architecture principles and vision.

Furthermore, it is hoped that the Open Source community will become involved in the use and extension of this SW Architecture. These users fall also into the category of developers. Documentation of the components will become a priority to ensure the success within this group of users.

2.4 General Constraints

As this is a generic architecture there are, in theory, no constraints on the types of objects that can be put into this system for retrieval (video, documents, who knows what in the future?) and no constraints on the ways in which an object can be viewed. As we are still looking for this Holy Grail of Generic Information Retrieval, this document will take a narrower view and will abide by the following general constraints:

- The only Information Resources to be considered will be document collections.

- Time for designing the Representation/Resource Domain SW Architecture is limited to 16 weeks.
- The basis for the SW Architecture will be an Open Source environment.

The entire Generic Information Retrieval SW Architecture will be based on the concepts and components available in the Open Source Software community. This enables third party involvement such as MSc students, programmers from interested software companies and the open source community in general. This means that eventual code and/or documentation will have to be made available to the open source community, either by maintaining it locally or turning it over to part of the open source organization.

3 Specific Requirements

3.1 Functional Requirements

Functional Requirement 1: Add New Resource Collection.

Introduction This service is to provide for the addition of new Resource Collections to the Generic Information Retrieval System. This will involve the use of a RAM (Resource Access Mechanism), a generic description covering the access mechanism that deals with the Resource Collection storage implementation specifics. In the scope of this document we will be assuming that our Resource Collection is a document collection.

Input Adding a Resource Collection will entail providing a RAM to access the Data Sources within the new Resource Collection. This RAM will most likely be resource specific.

Processing The Resource Collection will be accessible by a RAM. This will give standard access to the Data Resources within and provide a level of abstraction with regards to the physical storage. The RAM is therefore responsible for a normalized view of the Resource Collection. This is the first step in the process of providing a normalized view to the Representation Domain and eventually to provide the Searcher with an answer to its query.

Output The Generic Information Retrieval System will have access, through a RAM, to the information contained in the Data Resources of the new Resource Collection.

Functional Requirements 2: Modify Resource Collection Scheme Definition.

Introduction This is a change to the description of a Resource Collection, something that can be seen as an external event for these requirements. This is a change made outside of the system being analyzed, but this will have an effect within the system. Changing a Resource Collection Scheme Definition means that the meta-data with regards to the Resource Collections content has been adjusted.

This leads to the question as to when changes to an existing Resource Collection will be propagated to the system. The change of a Resource Collections population will be covered in 7.3. We will consider the change to a Resource Collection Scheme Definition as being separate from the population of the change, but this requirement will most likely initiate a chain of events from requirement 2 to requirement 3 to reflect the changes.

This service is to provide for the changing of an existing Resource Collection Scheme Definition. This will allow for addition of new extensions to the way information within an Resource Collection can be organized for external access. A Resource Collection Scheme gives meaning to the contents within the collection by providing a mapping of the available Aspect and domain information contained within the Data Resources within the Information Resources that are in the Resource Collection.

Input A modification has taken place in the Resource Collections Scheme Definition, a change in Aspect and domain information.

Processing The modified Aspect and domain information will be made available to the Normalization process, so that the Generic Information Retrieval System can be kept up to date with respect to available Information Resources. This update mechanism will be maintained within the Representation Domain, which means that the Normalization process will be responsible for updating changing Resource Collection Scheme Definitions. The frequency will depend on the time needed to update the population of the Resource Collection to reflect this change in Resource Collection Scheme Definition.

Output The new Aspect and domain information reflecting the change to a Resource Collection Scheme Definition has been made available to the Normalization process. The Representation Domain may or may not reflect this change, depending on the frequency chosen to update such a change. This output will lead to a chaining effect to the following requirement described in section 7.3.

Functional Requirements 3: Modify Resource Collection Population.

Introduction This service will allow for the changing of the contents of a Resource Collection, that is the Information Resources within an existing Resource Collection. To remain flexible, this could follow the changing of a Resource Collection Scheme Definition, see section 7.3. It is also possible that this can proceed a modification of a Resource Collection Scheme Definition and therefore require a change in a Resource Collection Scheme Definition. In the later case, this requirement will be followed by the application of Modify Resource Collection Scheme Definition.

Input New Data Resource(s) are added to existing Information Resource(s) within a Resource Collection. Alternately, new Information Resource(s) are added to a Resource Collection.

Processing This change in content will need to be made available to the Normalization process, this can be achieved by chaining the requirement Modify Resource Collection Scheme Definition to a change in content. Therefore the content change will be followed by an update to the Aspect and domain information (Resource Collection Scheme Definition) and then made available to the Normalization process. it will remain the Normalization processes task to update its own information with regards to available Aspect, content and domain information.

Output The new Data Resource(s) and Information Resource(s) in the modified Resource Collection are available to the Searcher.

Functional Requirements 4: Add New Aspect Domain Definition.

Introduction This service is to provide for the creation of a new Aspect Domain Definition, one which will give the initial view of an Information Resource to the system. This definition will describe how the Data Resources within an Information Resource present their content (i.e. physical structure) to the system, what the domain is (i.e. postscript, text, XML) and the Aspect to be made available (i.e. full-content, abstract, keywords).

Input At least a minimum (enough to make an Information Resource available to the system) Aspect Domain Definition will be supplied. Furthermore, the Information Resource must be in the system for the new Aspect Domain Definition to work with and there exists no Aspect Domain Definition for this Information Resource.

Processing The provided Aspect Domain Definition will describe the content, domain and Aspect to be provided by the Data Resources within the new Information Resource. It is the responsibility of the Aspect Domain Definition to provide a Normalization for this new Information Resource.

Output A normalized view of the new Information Resource(s).

Functional Requirements 5: Add Translation Definition.

Introduction This service is to provide for the addition of new Translation Definitions from one Aspect Domain to another with the results that a new Aspect Domain becomes available within the system. This will lead to new Data Resources from existing Data Resources.

Input There exists an Aspect Domain Definition that provides a representation of a Data Resource within an Information Resource. A new Translation Definition is provided that maps from this existing Aspect Domain to a new Aspect Domain.

Processing The existing Aspect Domain provides the basis for a mapping from one Aspect Domain to another. The Translation Definition will provide the mapping function, which will also ensure that the newly mapped Aspect Domain is made available next to the existing Aspect Domains.

Output There exists a new mapping from one Aspect Domain to a new Aspect Domain, resulting in new Data Resources for the Searcher.

3.2 Non-Functional Requirements

Non-Functional Requirement 1: Heterogeneity.

The Generic Information Retrieval SW Architecture should not be limited to any specific platform. The initial setup will be as an Open Source SW environment to promote the continued development and usage by as wide an audience as possible. Therefore the Generic Information Retrieval System will make use of existing standards with regards to networking, communication protocols, storage, query languages, Object Orientation and existing Open Source software.

Non-Functional Requirement 2: Evolvability.

The Generic Information Retrieval SW Architecture should be responsive to the future development of new Information Retrieval technologies and research needs. It should be open to new needs, new services and new facilities that will require an environment to provide inter-operability between these new technologies. This is a more important aspect than efficiency and quality of service.

To achieve this will require that the components used have to be very self containing with well defined interfaces to their environment. The most logical method to ensure this is to think in Objects and Services to be provided within the Generic Information Retrieval SW Architecture. This will lead to the separation of functionality and allow for the most flexibility.

Non-Functional Requirement 3: Inter-operability.

The Generic Information Retrieval SW Architecture should support the ability to interconnect and communicate with various components in various implementations. The various implementations of Information Resources (databases, collections, objects) and the components that will communicate with them will need to support a wide variety of inter-operability. This will be achieved with the use of distributed computing concepts such as middle-ware, standard Internet protocols, standard query languages, standard networking interfaces and well defined interfaces between component layers.

For example, the Resource Access Mechanism will be left out of the scope of this architecture to provide for inter-operability. By stating that the RAM must provide communication over a network using standard protocols and query languages, we force this part of the architecture to maximize its inter-operability. The components from within the architecture will only need to maintain communication via the standards mentioned to inter-operate with the various RAM's.

Non-Functional Requirement 4: Extensibility.

The Generic Information Retrieval SW Architecture will need to provide a stable core architecture to support the basic integration and inter-operation services needed by Information Retrieval system developers and researchers. On top of this, it will need to be extensible for future development, usage and research projects that may vary in the direction they take within Generic Information Retrieval research. Currently this architecture is only directed at making information available (supply), but eventually in the future it will have to take into account the request for information (demand).

The entire Generic Information Retrieval SW Architecture will be set up to facilitate easy system development for AIO research prototypes. There will be a need for well defined interfaces between components from the different layers of the Generic Information Retrieval System to facilitate the addition of new components. This can be achieved by applying available standards such as CORBA, ORB and Java to name but a few. Once again the goal is to remain flexible but still achieve the goal of prototyping for Generic Information Retrieval Research.

Non-Functional Requirement 5: Performance.

Due to the possibility of very large amounts of data in a Resource, the computational power needed to process this will become a performance issue. The Generic SW Architecture should provide for a basis with which to conduct the desired Generic Information Retrieval research projects and provide continued performance as needed in the future. The Generic Information Retrieval SW Architecture will try to use existing building blocks for the storage and retrieval of data to provide satisfactory (and proven) performance.

Non-Functional Requirement 6: Security.

The information being transported could conceivable contain sensitive data that needs to be protected. These issues will be dealt with using existing open security standards to provide the best security available over time. This is in line with the open standards mentioned in other non-functional requirements.

3.3 External Interface Requirements

There will be requirements as to this part of the Generic Information Retrieval SW Architecture, but as of now they are outside of the scope of this project.

7.4 Appendix D - GNU Free Documentation License

GNU Free Documentation License
Version 1.1, March 2000

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the works title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices and the license notice saying this License applies to the Document are reproduced in all copies and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100 and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five). C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section entitled "History" and its title and add to it an item stating at least the title, year, new authors and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. In any section entitled "Acknowledgments" or "Dedications", preserve the sections title and preserve in the section all the substance and tone of each of the contributor acknowledgments and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version. N. Do not re-title any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Versions license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgments" and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection and distribute it individually under this License, provided you insert a copy of this License into the extracted document and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate" and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Epilogue

Long ago the Navajo Indians in the south west of America created pottery, often functional pieces that would be used to carry water or store food. The potters would decorate their creations with painted drawings that were unique to each creator. When one views a clay pot made by a Navajo it should be noted that the pattern runs around the entire jar, but never completes a circle. This has a special meaning to the Navajo, they believed that the pot was never completed just as the potters life was not yet completed. There is always a break in the circle, a break that implies continuation. Therefore, as the Navajo potters never complete the circle on their pots, I too end this part of my studies with a semi-colon to indicate that I hope to continue learning from the world around me;

Bibliography

[Amyot99] D. Amyot, *Use Case Maps Quick Tutorial*, UCMtutorial.pdf pp. 15-16, SITE University of Ottawa, Sep. 1999,
http://www.usecasemaps.org

[Z39.50] ANSI/NISO Z39.50-1995, *Information Retrieval(Z39.50): Application Service Definition and Protocol Specification*, Maintenance Agency Office, July 1995.

[Bass98] L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice*, Software Engineering Institute, pp. 25, Addison Wesley Longman Inc., Boston, MA., 1998.

[Bruza90] P.D. Bruza, Th.P. van der Weide, *Two Level Hypermedia - An Improved Architecture for Hypertext.*, Proceedings of the Data Base and Expert System Applications Conference (DEXA 90), pp 76-83, Vienna, Austria, 1990.

[Bruza93] P.D. Bruza, *Stratified Information Disclosure - A Synthesis between Hypermedia and Information Retrieval*, Thesis Publishers Amsterdam, pp.20-23, Amsterdam, 1993.

[Czarnecki00] K. Czarnecki, U.W. Eisenecker, *Generative Programming*, Chapter 4, Addison-Wesley, Boston, MA., 2000.

[Crawley99] S. Crawley, *CORBA[tm] Meta Object Facility*, DSTC Report to Industry, Brisbane, Australia, October 1999.

[deBruin01] H. de Bruin, H. van Vliet, *Scenario-Based Generation and Evaluation of Software Architectures*, Free University, Amsterdam, 2001.

[Eliens95] A. Aliens, *Principles of Object-Oriented Software Development*, Chapter 1, Addison-Wesley, Boston, MA., 1995.

[FreeDB02] Linux SQL Databases and Tools website, *Free database listing*, 2002,
http://linus.org/linux/db.html, 2002.

[Hofstede96] A.H.M. ter Hofstede, H.A. Proper, Th.P. van der Weide, *Query Formulation as an Information Retrieval Problem.*, The Computer Journal, Vol. 39, pp. 225-274, 1996.

[IEEE98] *IEEE Recommended Practice for Software Requirements Specifications.*, IEEE Std 830, 1998.

[IRISweb] Information Retrieval Information Systems Group, *IRIS homepage and mission statement, 2002,* http://www.cs.kun.nl/is/index.html

[Jini02] Sun Microsystems, *Jini[tm] Network Technology*, 2002, http://www.sun.com/software/jini

[Klein99] M. Klein, R. Kazman, *Attribute-Based Architectural Styles.*, Carnegie Mellon Software Engineering Institute, CMU/SEI-99-TR-022, October 1999.

[Kzaman00] R. Kazman, S. J. Carriere, S. G. Woods, *Toward a Discipline of Scenario-Based Architectural Engineering.*, Annals of Software Engineering, Vol. 9, 2000.

[mysql02] MySQL website, *MySQL - The worlds most popular open source database*, 2002, http://www.mysql.com

[Ockerbloom98] J. Ockerbloom, *Mediating Among Diverse Data Formats*, Carnegie Mellon University, CMU-CS-98-102, January 1998.

[Papazoglou01] M.P. Papazoglou, H.A. Proper, J. Yang, *Landscaping the information space of large multi-database networks.*, Data & Knowledge Engineering, 36(3), pp. 251-281, 2001.

[postgresql02] Postgresql website, *PostgreSQL*, 2002, http://www3.us.postgresql.com

[Potts94] C. Potts, K. Takahashi, A. Anton, *Inquiry-Based Requirements Analysis.*, IEEE Software, pp. 21-32, March 1994.

[Proper99] H.A. Proper, P.D. Bruza, *What is Information Discovery About?*, Journal of the American Society for Information Science 50(9), pp. 737-750, July 1999.

[Proper01] H.A. Proper, *Profile based retrieval of networked information resources (PRONIR)*, research proposal, 2001.

[SEI02] Carnagie Mellon Software Institute, *Domain Engineering: A Model-Based Approach*, 2002, http://www.sei.cmu.edu/domain-engineering

[Vliet93] H. van Vliet, *Software Engineering - Principles and Practice.*, J. Wiley, West Sussex, England, 1993.

[Wondergem00] B.C.M. Wondergem, P. van Bommel, T.H.P. van der Weide, *Matching Index Expressions for Information Retrieval*, Information Retrieval, Vol. 2, pp. 337-360, 2000.