

Semantic Agent Support for Managed Open Information Retrieval Services

Xuan Huang

Submitted for the degree of Doctor of Philosophy

Department of Electronic Engineering

Queen Mary, University of London

2007

Declaration

The work presented in the thesis is the author's own.

DATE: _____

SIGNATURE: _____

To my beloved father and mother

Abstract

Distributed open information retrieval involves posing queries that may span across multiple data resources and involve users with multiple levels of understanding, presenting a challenging problem domain with multiple levels of data heterogeneity. To ease data interoperability and to tolerate the data heterogeneities, a semantic mediation approach is used to capture the key domain concepts and the relations between them. Semantic mediation processes naturally support query transparency, masking the underlying data model complexity, enabling users to specify what to query at a higher level of data abstraction, rather than having to specify where or how to query at a low-level. Within the context of the EU Environment Data Exchange Network for Inland Water, EDEN-IW, research project, a distributed semantic model represented in the DAML+OIL Ontology Language, has been created to enable queries to be mediated across multiple heterogeneous database resources. A flexible and robust Multi-Agent System (MAS) framework has also been developed using the JADE Agent toolkit to provide services to share, access, mediate and manage a distributed and partitioned IW Ontology model, and to synchronise it with any corresponding dynamic non-semantic data resources.

As part of this project, a decentralised directory service has been developed to improve the availability of metadata repository, so that if the central directory is unavailable, distributed repositories can take over service discovery. Service quality metadata obtained from service invocation sessions is incorporated into the metadata model for service providers so that service discovery is then not only based on service capabilities but also on the quality of service. Secondly, an introduction service that avoids third-party directory mediation is pioneered to initiate communication between agents. Thirdly, a semantic virtual data warehouse service has been developed to allow users to flexibly aggregate the results of queries. Finally an exception handling model has been developed to provide fault-tolerance for query related exceptions in open information service environments as part of the robustness solution of the framework. The semantic context of the interaction is leveraged to optimise the exception handling. Two major exception handling methods are investigated, one is the use of substitutes to resume query

processing from total agent failures, and the other is the use of the semantic mediation method to mask and handle partial agent failures.

Acknowledgements

I would like to take this opportunity to thank my supervisor, Dr. Stefan Poslad, for his grand visions that guided the research project throughout and his incredible patience with me. My gratitude also goes to the Head of Department, Prof. Laurie Cuthbert, for his firm belief in my ability to finish the PhD and his help along the way.

A big thank-you to the wonderful staff of the Department of Electronic Engineering at Queen Mary, whose help over the years has been invaluable. In addition, I would like to thank all the colleagues of the EDEN-IW project, working with them had been a delightful experience.

Also not forgetting the friends who made this journey much more enjoyable, Athen, Bin, Bob, Chengmiao, Choo, Chris, Costas, Dave, Dejian, Handong, Hengzhi, Huifang, Jianxin, Jim, Jimmy, Jin, Karen, Landong, Leonid, Lester, Lin, Ling, Matt, Na, Ning, Qiang, Rupert, Terry, Tom, Vindya, Weekian, Weizhi, Xu, Xuan Jin, Xuefei, Yapeng, Yasir, Ying, Yiran, Yong, Yue Chen, Yue Gao, Zekeng, Zhijia, and Zhongning.

Finally, my love and gratitude go to my parents. My dad had always believed in me, and my mom's love and encouragement are always there for me. Without them, this thesis would not have happened.

Table of Content

Table of Content	7
Table of Figures	10
1 Introduction.....	14
1.1 Summary of Problem Properties.....	18
1.2 Research Objectives.....	18
1.3 Thesis Outline.....	19
2 Hybrid Distributed Information Retrieval Systems based upon Multi-Agent Systems.....	22
2.1 Distributed Information System.....	22
2.1.1 Distributed Information System Architecture.....	23
2.1.2 Information Retrieval Processes.....	24
2.1.3 Complexity of Information Queries.....	25
2.1.4 Requirements.....	27
2.2 Data Integration versus Data Interoperability.....	28
2.2.1 Data Warehouses.....	28
2.2.2 Physical vs. Logical Data Warehouses.....	29
2.2.3 Schematic vs. Semantic Integration.....	30
2.3 Distributed Query Management Services.....	31
2.3.1 Metadata Management and Directories.....	31
2.3.2 Heterogeneous Data Interoperability.....	33
2.3.3 Query Orchestration and Data Aggregation.....	33
2.3.4 Derived Data Analysis.....	34
2.3.5 Fault-tolerance for Query Management.....	34
2.4 Data Interoperability Architectures.....	36
2.4.1 Distributed Databases.....	36
2.4.2 Client-server Web Service Models.....	37
2.4.3 Data Grids.....	38
2.4.4 Semantic Web.....	38
2.4.5 Intelligent Information Agents and Multi-Agent Systems.....	39
2.4.6 Architecture Choice: Hybrid MAS.....	47
2.5 Summary.....	48
3 Literature Survey.....	49
3.1 Scope of the Survey.....	49
3.2 Semantic Distributed Information Retrieval.....	49
3.2.1 Multi-Agent Systems.....	49
3.2.2 Semantic based RDBMS.....	55
3.3 Directory Service and Service Discovery Mechanisms.....	57
3.4 Fault-tolerance for Open Services.....	59
3.4.1 Exception Handling in Distributed Artificial Intelligence.....	59
3.4.2 Exception Handling in Autonomic Computing.....	63
3.5 Discussion.....	64
3.6 Summary.....	66
4 EDEN-IW MAS to Integrate Distributed Databases.....	67
4.1 Inland Water Quality Data Domain.....	67
4.2 Information Integration Infrastructure: MAS.....	68
4.2.1 Architectural Overview of EDEN-IW.....	68
4.2.2 MAS Design Methodology.....	70

4.2.3	MAS Component Design	72
4.2.4	Metadata Structure	74
4.2.5	Agent Interactions	77
4.3	Distributed Query Orchestration	80
4.3.1	Task & Result Sharing Mechanism.....	80
4.3.2	Service-Action Model	82
4.4	Use Scenario	84
4.5	System Performance Measurement.....	85
4.5.1	Descriptions of the Experiments	85
4.5.2	Measurement Results	86
4.6	EDEN-IW Prototype Achievements	87
4.6.1	Author's Contribution	87
4.7	Discussion	89
4.8	Summary	90
5	EDEN-IW MAS Extension 1: Decentralised Directory and MDDV.....	92
5.1	Introduction	92
5.1.1	Application Scenario	93
5.1.2	Architectural Overview	95
5.1.3	Data Metadata and DSS Queries.....	97
5.1.4	Design Issues.....	98
5.2	Directory Service (Version2).....	102
5.2.1	Ontology Model	103
5.2.2	Internal Agent Design	107
5.2.3	External Agent Interaction	110
5.2.4	Matching Services based upon both Capability and QoS.....	114
5.2.5	Service Discovery	115
5.2.6	Maintenance of the Local Metadata Repositories.....	115
5.3	Semantic Multiple Dimensional Data View Design	116
5.3.1	DSS Data Queries	118
5.3.2	Multi-Dimensional Database Structure.....	119
5.3.3	Query Decomposition and Post-processing	121
5.3.4	Semantic Model for Data Analysis Application	123
5.3.5	View Templates	127
5.3.6	Constraint Modelling	128
5.3.7	Process for View Instantiation and Example	130
5.4	Evaluation of the Framework.....	131
5.4.1	Descriptions of the Experiments	131
5.4.2	Comparison of Different Service Discovery Methods.....	131
5.4.3	Data Integration and Harmonisation using Semantic Views	132
5.5	Discussion	134
5.6	Summary	136
6	EDEN-IW MAS Extension 2: Fault-Tolerance	137
6.1	Introduction	137
6.1.1	Problem Description	138
6.1.2	Exceptions and Failures	140
6.1.3	Architectural Overview.....	142
6.2	Characteristics of a Robust MAS in Handling Information Queries	143
6.3	Design Issues.....	143
6.3.1	Modelling of Goal Tasks and Plans	144
6.3.2	Finite State Transition Models for Agent Interaction Protocols	145

6.3.3	Exception Detection & Fault Diagnosis.....	147
6.4	The Exception Handling (EH) Framework.....	148
6.4.1	EH Agent and Exception Handling Workflow.....	149
6.4.2	Monitor and Re-planning Functions for Agents.....	150
6.4.3	Semantic Model for Exception-Fault Relations and Process Description.....	151
6.4.4	Heuristic Knowledge Modelling.....	153
6.4.5	Metadata Assisted Recovery Mechanism.....	154
6.5	Evaluation.....	156
6.5.1	Total Directory Agent Failure (Recoverable Failure).....	156
6.5.2	Total Task Agent Failure.....	157
6.5.3	Partial Resource Agent Failure (degradable failure).....	160
6.5.4	Description of Experiments.....	160
6.6	Discussion.....	163
6.7	Summary.....	164
7	Conclusion and Further Work.....	166
7.1	Contributions.....	166
7.2	Conclusion.....	167
7.3	Further Work.....	169
7.3.1	Agent Social Acquaintance Model.....	169
7.3.2	Orchestrate Services to Service Level Agreement.....	170
7.3.3	Enhance Fault-tolerance with Probabilistic Risk Analysis.....	170
8	Author's Publications.....	172
9	Bibliography.....	173
10	Appendix: Experiment Output for the Fault-tolerance Framework.....	185
10.1	Experiment one: DA failure.....	185
10.2	Experiment two: RA failure.....	188

Table of Figures

Figure 1 Schematic diagram of the development of fault-tolerant, flexible distributed IR MAS developed in three phases described in chapters four, five and six.....	21
Figure 2 Schema architecture of a layered heterogeneous database system.....	24
Figure 3 MAS properties: agents, derived from [55].....	41
Figure 4 MAS properties: system, derived from [55].....	41
Figure 5 Overview of agent communication in which one agent requests information from another	46
Figure 6 Key elements that affect water quality	67
Figure 7 Architectural overview of the EDEN-IW prototype system.....	69
Figure 8 MAS role model for EDEN-IW system	72
Figure 9 Overview of conceptual data model of an observation	76
Figure 10 The EDEN-IW Semantic data model of core Inland Water Concepts interlinks the different parts of the distributed information system: Presentation logic or application processes, e.g., Decision Support System (DSS); Resource management processes such as Semantic Mediation and the local database interface models (DB1-3).....	76
Figure 11 Summary of an agent registration message from a Task Agent to the Directory Agent.....	79
Figure 12 Summary of RDF fields in a UC1 query	80
Figure 13 Agent interactions for metadata queries and agent registration and service discovery processes.....	83
Figure 14 Agent Interaction for user query and metadata query	84
Figure 15 A plan for a user to retrieve a specific set of data	94
Figure 16 Architectural overview of an extended MAS to support an improved directory service and MDDV.....	96
Figure 17 Key concepts in the conceptual layer of the adapted Gaia model.....	100
Figure 18 Ontology for service quality and performance measurement.....	104
Figure 19 A process model of the Directory Service profile	105
Figure 20 An example of a rule to automate generation of queries.....	107
Figure 21 Data flows and control flow between modules of an agent.....	108
Figure 22 A Hello Protocol in AUML [2]	112

Figure 23 Dimension relations and query examples	120
Figure 24 Overlaps of ontologies	123
Figure 25 Three layer semantic model: multi-dimensional view ontology, IW global ontology and IW local ontology of IOW	125
Figure 26 Ontological bindings for the observation View template	128
Figure 27 Constraint of region "north-east" modelled using OWL-S.....	129
Figure 28 Workflow for SQL generation using views.....	130
Figure 29 User utility measurements of different service discovery mechanisms	132
Figure 30 Graph of observation values for Nitrate in river SEINE during year 2000	133
Figure 31 Observation values across three rivers	133
Figure 32 Query forming and analysis workflow with potential exceptions modelled in Petri Net [88].....	139
Figure 33 Agent's query processing workflow with exceptions in Petri Net [88].	140
Figure 34 Architectural overview with added exception handling agent and exception handling ontology	142
Figure 35 An example of task decomposition in HTN	144
Figure 36 Message processing transition model in Petri Net.....	146
Figure 37 State transition model for FIPA-Request initiator in Petri Net	146
Figure 38 State transition model for FIPA-Request responder in Petri Net	147
Figure 39 Generic exception handling workflow	149
Figure 40 Exception-Fault Ontology	152
Figure 41 Service description and process model in OWL-S	152
Figure 42 A rule in JESS that shows the change of the current goal.....	154
Figure 43 Interaction to identify a substitute	159
Figure 44 Interaction to resume query processing conversation.....	160
Figure 45 Hello message exchange between TA and RA.....	185
Figure 46 TA reasoning process	186
Figure 47 RA reasoning process	186
Figure 48 DA broadcasts announcement	187
Figure 49 TA reasons about what to do when directory is not available.....	187
Figure 50 TA broadcasts for a specific resource agent.....	188
Figure 51 Exception handling steps in RA	189

Figure 52 Reasoning in EHA upon receiving RA's exception report	189
Figure 53 Agent interaction in handling RA exception	190

Table of Tables

Table 1 Summary of surveyed project limitations comparing to desirable features..	64
Table 2 System components and their functionalities	72
Table 3 Agent functionalities & activities	73
Table 4 Results of 5 times repeated UC1 query. Times in milliseconds.	86
Table 5 Results of 5 times repeated UC9 query. Times in milliseconds.	86
Table 6 An example of the service quality metadata	105
Table 7 An example of metadata information on agent interaction.....	109
Table 8 An example of metadata on message content	109
Table 9 Aggregated Interaction Metadata.....	113
Table 10 Metadata about interactions for conversation 001	114
Table 11 Dimensions with levels and instance examples	120
Table 12 River data quality classification.....	121
Table 13 Table mapping between global, NERI and IOW	126
Table 14 Direct mapping of fields related to river from MDDVO to IOW DB	126
Table 15 Different ranking results	134

1 Introduction

Information interoperability across distributed heterogeneous data sources presents a challenging problem domain with multiple levels of heterogeneity and complexity. Consider a sub-domain of the environmental information management, that of managing European wide information about Inland Water (IW) quality [49]. Typically, alphanumeric data to aid the management of inland water is stored in multiple Relational DataBase Management Systems (RDBMS). RDBMS queries and operations are typically expressed in a rich standard data query language called SQL (Structured Query Language) [121]. Each RDBMS within the IW domain is designed in isolation to store its own unique data structures and to answer its own specific queries that relate to those structures. Hence the same type of data is likely to be stored in different structures across RDBMS. RDBMS with geospatial extensions are often used in order to relate the water quality values to geographical regions in the form of map images.

Different stakeholders naturally hold different user viewpoints about the domain. A policy maker might be interested in knowing if specific water framework regulation has had any positive results as evidenced through monitoring the water quality in different countries. Citizens may be concerned if their water quality is safe to drink and safe to wash with. Scientists may be interested in understanding if the trends of measurements support some specific theory they hold. Often a semantic gap exists between different stakeholders in terms of their user view of the data and their understanding of the finely grained and complex stored data structures.

Information queries may need to be spread across multiple related databases in order to perform aggregations and comparisons of water quality measurements across space, time and different types of water quality indicators. The individual databases have been created and are managed, autonomously resulting in the same types of information often being represented in different formats, expressed using different vocabularies and stored in different data structures. In this case, users may need to understand the low-level data structures of each of several different individual data sources and know how to orchestrate multiple processes to gather results from them

in order to correlate them. These types of cross database queries represent a new additional type of database application in addition to the individual database query applications oriented to querying and modifying a database in isolation.

A useful property of a distributed information system to simplify access is information resource virtualisation, i.e., to provide a global view of some common generic characteristics for the individual resources, such as location, time and type of water quality indicator. These are derived from explicit and tacit knowledge about the data models. These mappings between these common characteristics and individual resources are kept. Mappings vary across the individual database resource instances because of the heterogeneity of the data model supported and the varying database system vendor support for SQL. Database resource virtualisation enables users to query data based upon what rather than where, using the generic characteristics as query constraints without having to know where the individual data resources are located and how they are structured. Essentially virtualisation adds a level of indirection to data access but it increases the complexity in having to maintain a set of data characteristics in the form of high level indices and their corresponding mapping to the low-level data instances.

Managing distributed queries is complex because it may require a decomposition process in order to determine how to convert a single query into the individual queries that can be directly answered by each autonomous database resource. Coordination of multiple individual query processes is needed in order to gather and combine the individual results into a meaningful whole. Data queries may need to be mediated to harmonise the results as these may be expressed using different data structure and different measurement units and different terminology. Coordination of distributed queries must weigh up how to deal with individual data resources intermittently generating huge quantities of data in response to particular queries and taking non-deterministic amount of times to return their results. Policies need to be defined to balance returning partial results earlier versus returning fuller results later.

In an open distributed information retrieval system, the status quo and loose autonomy of the individual information resources is maintained rather than tightly organising them into a rigid structure. This is because the individual RDBMS have

autonomous owners and their primary purpose is to answer individual queries rather than to answer cross database queries. Restructuring them to better answer cross database queries may require too much re-engineering and may be too disruptive. This open distributed system must take into account data resources and services behaving dynamically. They can come on-line and go off-line by themselves, new data can be added and old data can be removed. This makes the resource virtualisation and data location and data storage structure transparency more complex to maintain, as the mapping between the virtual resource data model accessed by the user and the actual stored data may need to be updated.

An essential enabler to support data resource virtualisation and to support cross heterogeneous database queries is metadata, data that describes data. However, there is often only minimum metadata that is explicitly modelled and that is available in an online computation form to describe the structure of data models and how they stored in the individual databases. Usually, the metadata contains no semantics to express: the meaning of the data in terms of the structures that define them; their relationships with related data structures and constraints for the data structure and how additional data can be derived or inferred from the data. Semantic metadata is needed to define the virtual data view and the mappings to the individual stored data including where the individual data is held. Semantic metadata is needed to define different user views of the data and to define mappings to the virtual data and hence to the individual data resources. Semantic metadata is needed to aid the coordination of the individual data processes to answer a query across multiple data resources.

The semantic metadata should be represented formally such as using an Ontology [4], for on-line use, to recall data instances with precision. Just as data requires quality data management processes to help prevent data anomalies, metadata also requires quality metadata management processes to execute the mapping and to check mappings to prevent metadata anomalies, e.g., the mapping between the virtual data and stored data may become incorrect because the stored data has moved or become updated.

Distributed information systems are challenging to manage. For example, individual data queries processes may fail, this in turn may cause cross data resource query

processes to fail. Data processes may encounter unexpected inputs due to data heterogeneity, communication links may intermittently break, data resources may become intermittently available and coordination processes may fail because the volume and time for retrieval may not be harmonised. Often metadata is stored and retrieved from a single centralised directory service. This represents a single point of failure for the system. If the director service becomes unavailable, the whole virtual data resource access model fails. Note often the directory stores metadata descriptions about resources and services from the perspective of the provider not from the perspective of the users who may have a narrower perspective of how the system to operate, e.g., of the Quality of Service that they perceived.

Fault-tolerance, the system's ability to supply regular service operations in the presence of hardware and or software failures is one of the key factors for improved system dependability. Two general approaches to handling non-catastrophic failures are either recovering the lost function with a substitute or resuming operation by correcting or bypassing the faulty process. A substitute is identified as an entity with the same functions of the failed entity.

Correcting faults to recover from failures is a challenge. Often fault-tolerance is designed to be handled locally for this reason error messages generated by faults are very low-level and often expressed at the level of operating system calls. Whilst, there are benefits in containing faults locally, this often may not prevent them from propagating to a more global application level because the application context of the fault may mean that a particular kind of fault handling should be selected. A second problem with local fault handling is that error manifestations may be caused by multiple faults and it may be unclear how to handle the error and correct the fault. For example, lack of results from a database server within in a certain time frame may lead to a time-out error. This may be caused by too high a volume of data being generated to return within the time, a broken network link or an unavailable server. Error messages can often be too low level to have a usable meaning at the level of the application processes that generate them. This makes it difficult for application processes to plan how best to assess the effect of a fault and how to change their operation to optimally or sub-optimally recover from them. The use of additional

semantic metadata about faults and errors could lead to application processes being better informed to handle faults more optimally.

1.1 Summary of Problem Properties

The key challenges for open distributed information retrieval and analysis include the following: firstly, results from multiple heterogeneous data resources must be accessed and aggregated whilst masking the complexity of query processing and alleviating users from needing to be familiar with the low-level data schema for each data resource accessed. Secondly, flexible data analysis is often needed in order to provide an abstract view of the data that users can understand: this requires the flexibility to create multiple view abstractions for analysis that caters for different users. Thirdly, in order to retrieve data, metadata such as data resource descriptions are derived from and synchronised to the data. The metadata is often searched instead of the data. However, the management of this metadata is complex. Finally, some levels of fault-tolerance are desirable for distributed information retrieval systems because of the various data heterogeneities and openness of the system that enables data and metadata resources to change whilst being synchronised properly with other dependent data.

1.2 Research Objectives

Based on the problem properties outlined in the previous section, the research presented in this thesis focuses on four specific objectives:

1. To investigate methods to:
 - a. Simplify access to distributed, autonomous, and heterogeneous data sources through resource virtualisation via global data views, at a higher level of abstraction in a minimally-disruptive fashion;
 - b. Coordinate query processes that may need to correlate or aggregate information across these.
2. To investigate use of a second layer of virtualisation to support multiple heterogeneous user viewpoints of data supported by a semantic metadata framework.

3. To investigate how to manage semantic metadata to better support 1 and 2. In particular to focus on user centred semantic metadata discovery mechanisms that go beyond using service preferences to query directories of service capabilities such as by incorporating and updating Quality of Service usage information. In addition, support for more distributed and fault-tolerant metadata directories is investigated.
4. To investigate the use of semantic propagation of low-level errors to better inform the application processes they are triggered in and thus to improve the fault-tolerance of distributed queries to heterogeneous information resources.

1.3 Thesis Outline

The thesis is organised into seven chapters including this one. Each chapter, apart from the first and last chapters, ends with a discussion and summary. Thus the reader can read the first and last chapter and the summaries at the end of each of the chapters, two to six, to gain a good overview of the thesis. The remaining chapters are organised as follows.

Chapter two starts by analysing the complexity of distributed information systems. It focuses a particular problem domain, the management of information concerning inland water quality. Based upon this analysis, a set of six main problem properties are identified and of these four are used as the problem requirements for the scope of this thesis. Based upon these requirements, the basic design choices for a framework to research and solve these problem requirements are introduced. The main focus is on an analysis of data interoperability rather than data integration frameworks. Based upon this, an architectural choice of a hybrid architecture is proposed that uses a Multi-Agent System (MAS) [126], the Semantic Web [4] and a Virtual Data Warehouse [121].

Chapter three is a critical survey of the state of the art research. This reviews the related work. Its scope is to focus on experiences in developing and applying the architectural choice from chapter two to heterogeneous distributed information systems whose main information resources are relational database systems. After surveying projects and systems individually, these are compared using the problems

requirements identified in chapter two in order to identify best practice and the limitations of the state of the art.

Chapter four is the first of three chapters that define the development of an experimental framework that innovates beyond the state of the art given in chapter three to solve the problem requirements in chapter two. The experimental framework in chapter three focuses on the author's main contribution as part of the EU FP5 IST Environmental Data Exchange Network for Inland Water or EDEN-IW project [49]. The main focus of the EDEN-IW framework was to develop a basic MAS and semantic framework to support the distributed queries and interoperability requirement for multiple RDBMS resources.

Chapter five extends the basic system in chapter four that was limited because it used a provider oriented centralised semantic directory service and offered limited support for multiple user virtual views of the application domain. It is done by developing a decentralised directory service and virtual data warehouse service sub-framework and evaluation.

Chapter six focuses on adding a sub-framework to give fault-tolerant support to the distributed query framework that was developed in chapters four and five, and evaluating it.

The relations between the three main methodology chapters (chapter four, five, and six) are shown in Figure 1. The framework is developed in three stages: chapter four presents the basic distributed query processing support developed as part of the EDEN-IW project, chapter five describes the extension to support a decentralised directory service and multi-dimensional data views, and chapter six adds a fault-tolerance capability to the system with a dedicated exception handling agent and associated exception-fault ontology. Each of the methodology chapters has a separate architecture diagram highlighting the incremental development (see Figure 7, Figure 16, and Figure 34).

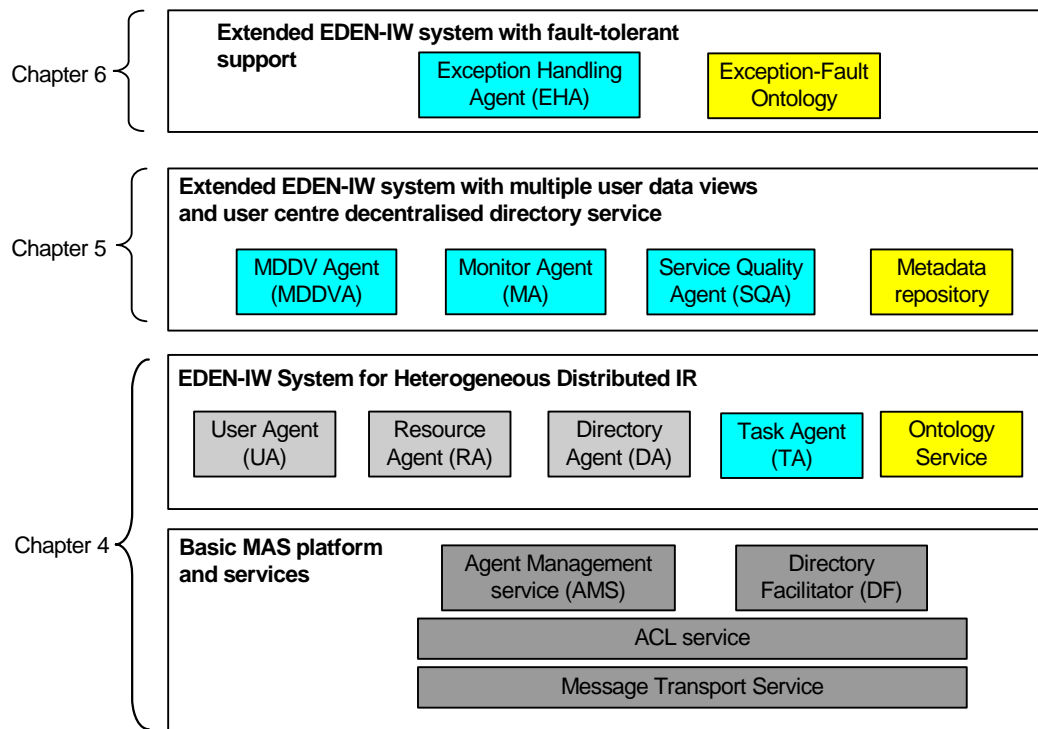


Figure 1 Schematic diagram of the development of fault-tolerant, flexible distributed IR MAS developed in three phases described in chapters four, five and six.

Finally, chapter seven presents the main conclusions with research contributions, and discusses future work.

2 Hybrid Distributed Information Retrieval Systems based upon Multi-Agent Systems

Distributed information retrieval and integration is a very active research area. A general concern is how to reuse so called legacy data sources that are often designed to support very specific applications, to enable them to interoperate more dynamically with newer applications that are specified after the data sources structures were designed and implemented, e.g., to use a database originally designed to answer local queries to answer cross database queries. A second major challenge is how to support data interoperability versus data integration, see Section 2.2, when dealing with the need to interlink data from multiple autonomous heterogeneous data sources within the same domain. Finally, queries may need to be sent to multiple data sources and orchestrated in a transparent way, so that users do not have to be concerned with the structure of each low-level data sources. These issues are discussed in this chapter using the application of inland water environment information for illustration. The system ends by introducing and discussing the main design models that can solve these challenges based upon data warehouses, conventional client-server models, distributed databases, Grid utility models [3], Semantic Web [4] and Multi-Agent systems [126]. A case is made for selecting a hybrid MAS architecture to support data interoperability and orchestration.

2.1 Distributed Information System

An Information System in a broad sense refers to a system, automated or manual, that comprises people, machines, and or methods, organised to collect, process, transmit, and disseminate data. Here the information systems of interest are those with machine processable forms of data storage that support (semi-automated) processing of data. As one of the most dominating data storage, querying and management solutions, relational databases play a key part in modern information systems. However this relational model is facing challenges from increasingly complex applications that require interoperability of various data sources.

Demands on information interoperability increase when using cross-domain database resources and applications. Heterogeneous distributed information systems are characterised by, multiple levels of heterogeneity ranging from different backend data storages, different querying languages, to different granularities of data retrieval. In addition, the availability of data sources is reduced when there are many concurrent users, when large queries that return large record sets are being processed and when communication links become disconnected or congested. In addition to bridging distributed heterogeneous data sources, there is a trend to build open distributed information systems, meaning that information sources and information processing and presentation services can join and leave the system freely. The openness dimension introduces uncertainty in that information sources may disappear without any prior notice. It also means that extra measures on handling these uncertainties are necessary for the coherence of the system.

2.1.1 Distributed Information System Architecture

Information systems logically can be separated into three levels of functionalities: resource management layer, application logic layer, and representation layer. Resource management deals with different data sources of information system, independent of the nature of these data sources. The application logic layer deals with the data processing to reflect the particular business objectives and usage of the data. The representation layer interacts with the external entities to present the information to the clients. A schematic architecture is shown in Figure 2.

The data that can be automatically processed by the application system is at the syntax-level, or the form of data is not significant unless corresponding processing has been identified. That is to say, raw data by itself does not make much sense to human beings without appropriate processing mechanisms. Through the processing within a specific context, data becomes information, or something people can interpret. This is the semantic level. Finally, information become knowledge, when an individual knows and understands how, when and why to apply the information. This corresponds to the pragmatic level.

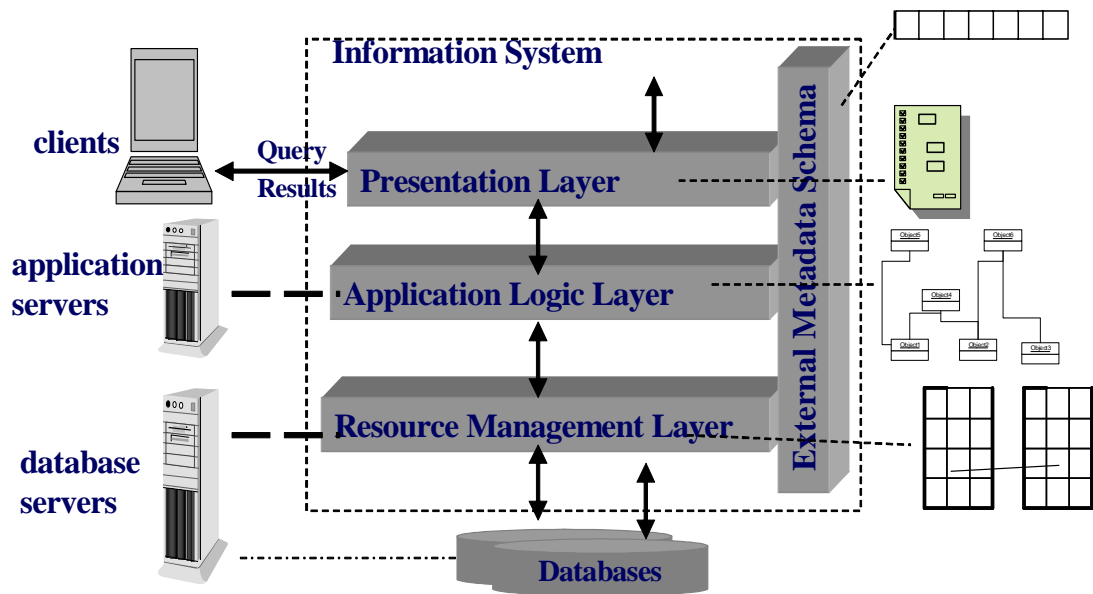


Figure 2 Schema architecture of a layered heterogeneous database system

2.1.2 Information Retrieval Processes

Data repositories need to be managed so that structured data can be efficiently stored and retrieved and so that access can be controlled and data quality can be maintained. The dominant model to support this is the Relational Database Models based upon access using the ANSI standard Structured Query Language (SQL). Relational DataBase Management Systems (RDBMS) organises data into tables that can be linked using key relationships to define one-to-one, one-to-many and many-to-many functional relationships between entities and uses SQL to create, read (or query) update and delete data. Note that even though this is a standard, it is a very complex standard, e.g., the SQL-3 specification published in 1999 consists of 2100 pages [121]. Note there are differences between RDBMS systems in the degree to which the supported standard and supported proprietary extensions, e.g., for metadata tables, access control, user-defined data structures etc. A key limitation of RDBMS systems is that there is no good model for searching and discovering data within a database and across databases and SQL does not define a standard way to exchange, import and export data between databases.

Information retrieval is not a simple process, which could be further decomposed into several tasks, including information acquisition, filtering, aggregation,

harmonisation, presentation and management. Retrieving information from distributed heterogeneous data sources is even more complex, in the case data from different sources need to be harmonised it consists of at least four main steps, i.e. locating the relevant data source, distributing information queries, formulating the right query that a specific data source can “understand” and finally applying harmonisation processing on results from different sources. That is the perspective of Information Retrieval as orchestrated processes or workflows, consisting of multiple interacting processes.

At the content level, Information Queries (IQ) concern how best to capture the semantics of the input and expected output. Content processing has to ensure that the query semantics is interpreted unambiguously so that different applications can interoperate. Metadata summarises the data sources and content, thus promotes browsing and searching of the data. From the system architectural point of view, IR connects two portals, one facing the data sources, and the other facing human users. The effectiveness of an information integration middleware is assessed with regard to the interaction at these two portals.

2.1.3 Complexity of Information Queries

An information query is complex due to the following reasons: the inherent difficulties in managing distributed system, in harmonising various data heterogeneities, and in bridging the gap between end users’ expectation with what is available in the data sources. These difficulties are further explained as follows:

- End users use higher-level terms and a high-level form based query, e.g., users may prefer to frame queries using terms such as dirty rivers but DBMS uses SQL queries using terms that link to the stored data structures. The concept dirty is not available in the stored data structures, however it could be defined as related to a collection of concentration values of some key determinands. By comparing the collective values to what Water Framework Directive defines as a threshold value, a water quality could then be judged to be dirty or not.
- Users are not aware of the SQL structure. SQL is a very specific language for DB administrators or system developers, while normal end users facing a

browser are most familiar with value-fitting form-based query interfaces. The developers of an information system cannot assume end users are able to master certain level of the SQL syntax to be able to construct the appropriate queries.

- Users / scientists need to find out where information is being held / being measured before they can query the right data source. It is the case without any middleware in between the DBMS and end users, provided that data sources are distributed and possibly anonymous for data protection, it would be extremely difficult for end users to locate sources before they can query. Middleware however masks the complexity, and it bridges the gap between users and data sources.
- Some queries may require information from multiple databases and data analysis support to answer it, e.g., to allow policy-makers to compare water quality data across Danish, French and English European rivers; to assess water quality in cross-border areas.
- Differences in data structures to store the same concepts can vary across different databases, e.g., a key inland water parameter in one database table may require a join between 2 tables in another database.
- The system inherently has no central control as data sources are autonomous and may become unavailable at different times - distributed failures are more difficult to handle.
- There may be non-deterministic delays in receiving a response, because the service provider may be busy and the request queue is big, or the communication link may suffer interruption and so on.
- Failure information may not be understood or may be masked from the query application. Failure messages in the information system are not always useful, in that end users are in no way interested in fixing the problem, or the failure information is often too low level for them to make any sense of it. Most of the time, failures are handled locally and masked from the application level. From the user's point of view, suggesting a relevant result in case of search failure is much more important to enhance their experiences.

2.1.4 Requirements

Six key query management services are needed to support open distributed data resource queries:

1. Distributed data resource access, management and data exchange;
2. Heterogeneous data interoperability;
3. Metadata and service directories management;
4. Query orchestration and data aggregation;
5. Derived data analysis;
6. Fault-tolerance for query management.

The data resource access service is conventionally achieved with SQL and data management is left to specific RDBMS. Service support for heterogeneous data interoperability was the focus of another related research project [130]. Services 3, 4, 5 and 6 are the focus of the research presented here. These directly related to the problem properties of the problem domain (see section 1.1). For example, to better support distributed query transparency, metadata for the data sources is needed so that relevant data sources can be discovered by browsing the metadata.

For these four services within the scope of this PhD project, the detailed requirements are as follows:

1. Provide flexible and robust access to metadata to enhance distributed data queries.
 - a. Service metadata is represented in a consistent format, and the storage and management structure is decentralised to overcome single points of failure.
 - b. The metadata in distributed repositories should be kept synchronised and consistent.
 - c. There is at least one alternative path to service discovery when the centralised directory fails.
 - d. Services are searched not only based on their capabilities, but on their performances as well.
2. Coordination of distributed information queries, answering and post-processing.

- a. Data queries are decomposed into sub-processes if necessary, and based on the constraints between sub-processes, tasks undertaking sub-processes are scheduled.
 - b. Post-processing includes combining results from different data sources, performing appropriate calculations or translations on data values, temporal projection, and aggregation along geographical dimensions.
3. Apply high level analysis on retrieved data to support decision making, for example compare results from different countries, and identify trends in water quality.
 4. Fault-tolerance: provide fault-tolerance to query related exceptions, e.g., handling faults to keep the system operational.

2.2 Data Integration versus Data Interoperability

There are two main approaches to combine data from multiple data sources: data integration and data interoperability. The chief difference between them is that physical data integration involves selected data being moved or copied from the individual data sources into a joint repository where they are processed whereas data interoperability does not involve creating a joint physical repository. There are pros and cons to both approaches. Creating a joint repository can make the analysis and processing of the joint data more efficient as data communication costs during the analysis are less. However, data in joint repositories requires substantial storage resources to store the data and may only get refreshed periodically, once a day so data may be somewhat stale. In addition, the data source owners may not wish to allow sub-sets of data to be copied perhaps because they then lack the control to monitor how their data is being utilised.

2.2.1 Data Warehouses

For some high level complex data queries that require harmonisation of data from multiple sources, the process of breaking down a query into smaller queries that data sources can handle and to apply harmonisation post-processing on aggregated results every time can be very time consuming and inefficient. Data Warehouses or DW

[121] are designed to facilitate high level data analysis, by having a physical table that stores amalgamation of data from different data sources. On-line Analytic Processing or OLAP [121] concerns the analysis of business operations with the intention of making timely and accurate analysis-based decisions. OLAP is typically used with a multidimensional database (MDDDB), a data management system in which data are represented by a multidimensional structure [121].

2.2.2 Physical vs. Logical Data Warehouses

Building a traditional data warehouse can be very time consuming, especially when the raw data set is huge, populating the amalgamation table can take a long time. Any update and modification of the table can be problematic as well. However it does provide additional facilities, such as data cleansing. One of the major concerns of simply putting data from various sources into a single table is that some data might be redundant. How to eliminate this and maintain data integrity is the challenge for traditional data warehouses.

An alternative to physically combining data stored in databases is to use views or virtual tables to integrate data at a logical level. The logic integration approach usually involves building middleware to provide direct connections between different data sources and different applications. This logic approach is also called a virtual data warehouse, because it provides exactly the same data access to external users without having the physical structure.

The virtual approach avoids the complexity of building up physically merged data tables and maintaining them. It is thus easier and quicker to build and provides the same access to data as a traditional one would. In terms of extensibility of the structure or the potential to accommodate new queries without introducing too many modifications, the virtual approach is better. Without the data cleansing facility, the virtual data warehouse normally has data only in their raw form. This means extra data processing might be necessary. Also the virtual approach would always have the most updated data, because there is no propagation for the update from data source to amalgamation table to take effect.

A virtual data warehouse represents a complex data interoperability challenge - not only must the metadata to support operational data queries from heterogeneous databases within a domain be aligned and mapped to data but this must also be performed for the multi-dimensional metadata. An interoperability approach based upon semantic mediation will need to be extended to support the multi-dimensional data interoperability.

It is worth mentioning that these two approaches are not exclusive to each other, a virtual method can be complementary to the physical one. In some enterprises, it may be necessary to have both at the same time. The virtual approach can also be seen as an augment to the traditional data warehouse, by providing a rapid solution to disseminate information.

2.2.3 Schematic vs. Semantic Integration

Most data integration systems using conventional data warehouse are based on schema alignment, meaning that there might exist a global schema that combines each local database's schema. Numerous syntactic mappings between the global and local schemas are needed for integration. The scalability of this schema-based approach has limitations, because of the number of possible heterogeneous schemas and the difficulty in normalising these syntactic mappings. In addition, the syntactical approach lacks the support of computable on-line representations of the metadata schema.

Alternatively integration based on semantics is more extensible and scalable. The core of the semantic approach is an ontological model that understands the differences in various data schema and provides the rules for mapping from one to the other. Mappings in this case are constraints of concepts, based on the meanings of them. This way of linking the concepts is more flexible to accommodate changes. This also means that new database schemas can be added into the model with fewer changes, while adding a new schema in the schematic model would probably mean recreating every mapping. Another major advantage of having a semantic model is the reusability across systems and subsequent interoperability it brings.

2.3 Distributed Query Management Services

This section studies the main features of the key aspects for managing distributed query services.

2.3.1 Metadata Management and Directories

Metadata, annotations of data, or data that describes data, aids data retrieval by representing the main characteristics of the data to enhance data searching, selecting and retrieval and analysis. These are enhanced because the amount of metadata is smaller and it can be quicker to retrieve and process than the data it refers to. There are many different types of metadata to support information retrieval including: indexes for locations on physical storage devices; data syntax that defines the structure of data such as the schema of the stored data; the descriptions that characterise stored data resources and data processing services and the semantic meaning of the stored data in terms of the properties of data and its relationships to other data within a domain.

The metadata needs to be specified, represented in a form that is accessible on-line, organised, created and managed, and linked to the data it refers to. Additional operations are needed so that whenever the data gets updated, any associated metadata gets updated. Typically, metadata is stored in special data files called directories and managed using directory or discovery services. Directory services are often used to support a model of loosely-coupled and indirect access. Access to providers can be more dynamic if users are designed to query directories to select providers at the start of each session or before accessing a provider. The availability of the Directories is therefore crucial to the clients. They need to be always accessible and available else, user requests will stall. The data structures used in directories vary from Java objects, XML data sheets, to RDF triples and RDBMS tables even. As to which specific structure is chosen, it very much depends on the usage of the metadata and its size. For example RDF triples supported named relationships in contrast to the class and aggregation relationships of Java objects. Therefore it is also one of the key design issues for a metadata management model to

select an optimal structure for the application. The directories provide public interfaces for services or agents to query the metadata.

The metadata management life-cycle is essentially a data management process and includes the processes of creation, query, update, maintenance, and deletion. Metadata is normally an inherent part of the legacy applications that created the data, i.e., the applications that create the database table content. Often, metadata is not in a declarative form for online use, for example the database schema that describes the database structure are represented using data graphs. Metadata needs to be transformed into a computational format, in this case in the form of a domain specific Ontology model.

[108] describe in more detail the three key types of meta data processes that were used: metadata processes are needed to create the metadata in an online form, metadata must be maintained and managed and processes are needed to allow applications to use different views of the metadata, for example to sample different narrower perspectives of data, at different levels of abstraction. The metadata schema, in the form of a Common core Ontology called the EDEN-IW Global View (EGV) Ontology is created manually through a dialogue between database owners and the metadata middleware developers. This is itself a complex set of sub-processes such as determining the scope of the Ontology, considering the reuse of existing Ontology models, enumerating important terms and defining class, properties, constraints and instances of concepts.

To enhance Ontology reuse for different types of applications and users, the Ontology is partitioned into different parts to support core data queries, multi-lingual support and post-processing such as decision support. A simple versioning system is used to allow the different parts to be updated and to signal that a new version may need to be used [108]. [130] describes the mapping processes in more detail that are needed to support different user views and different application use of the EGV model.

Typically, a core part of each agent, loads and parses the Ontology it needs on start-up. Data Resource agents in the system contain Web access and database access

support to allow metadata to be used as constraints to access database data via a Web interface. User agents in the system contain Web access support to allow users to indicate the constraints for queries that they make at a Web interface. The multi-agent system and processes to support Ontology use by applications are described in more detail in chapters four, five and six.

2.3.2 Heterogeneous Data Interoperability

Data in autonomous data sources that need to be related is often heterogeneous in terms of its terminology or naming, structure, semantics and usage and the RDBMS. Metadata is needed to represent the heterogeneity characteristics. Mapping or data alignment processes are needed to transform data from one representation into another. Data integration processes are usually associated with merging data into a common format for use. Data interoperability may also align data into a common or global format or one party may align data into the local format of the other party when two application processes interoperate. However the existence of a common global format is not essential to achieving interoperability.

2.3.3 Query Orchestration and Data Aggregation

Query orchestration inherently supports query abstraction to allow users to specify queries that hide the detailed characteristics of the individual data sources such as their location, structure and access permissions. Users specify what not where or how. The query orchestration service must map what to where perhaps by using metadata in the directory service. It needs to translate high-level data query constraints into low-level individual query constraints such as mapping a named geographical region into a set of discrete points or stations as a representation of a region. It may need to coordinate queries and define policies for whether or not a slow to respond data source should hold up data from a faster responding data source from being correlated. It may need to harmonise the value of results, e.g., so that a water contaminant concentration can be meaningfully compared even though they are expressed in different units.

2.3.4 Derived Data Analysis

Data repositories are often designed to strike a balance between minimising the space of the stored data versus minimising the data retrieval time. Part of this design involves the use of metadata which although increasing the storage space, greatly speeds up data searches and data retrieval. New data applications, driven by new requirements can arise after the data sources are designed and created. For example, the environment databases are created to store the environment data collected. Later applications can be defined and added to reuse the unprocessed data that can support trend analysis by time and space aggregating data. Other applications can be defined to support decision support, e.g., to report which environment areas have the lowest water quality or when they fall below a certain threshold. Rather than reengineer to stored data structures, a more flexible approach is to process the raw stored data and to dynamically derive the appropriate processed data to fit the application as needed.

Derived data is data that can be generated from other data using a series of defined data operations, e.g., data sets can be time-averaged or averaged over spatial regions. Derived data can also be defined by proprietary or business specific policies, e.g., contaminated water is classified as any water where the average deviation of the concentration of each heavy metal from a norm is above a certain level. Derived data is often used to provide high-level abstractions of the data for use by managers and end-users who do not want to be concerned with the low-level detail of data. If common derived data is frequently accessed and the additional storage costs are deemed to be acceptable, derived data can be stored to improve derived data analysis. The derived data needs to be synchronised to the data so that if the data changes, the derived data will get generated or updated.

2.3.5 Fault-tolerance for Query Management

Distributed systems consist of autonomous yet distributed entities that are connected to each other by a communication network. According to Burns and Wellings [12] there are four main sources of failures:

1. Inadequate specification of software,

2. Software design error,
3. Processor failure,
4. Communication error.

The first two failures can only be dealt with offline. They are due to human mistakes which naturally require human intervention to fix them. The latter two kinds of failures can be avoided by careful exception handling techniques, which are the focus of the fault-tolerance model here.

Services are used to describe the external functionalities of autonomous entities. Key concepts of fault-tolerance can thus be defined based on the notion of services. An exception is defined as the deviation from “normal” service behaviour. Exceptions have causes, and some researchers further define these causes. Intuitively identifying the faults for every exception situation is useful in exception handling. It is very difficult in practice. Because sometimes one exception event can be caused by different faults, while sometimes a fault can cause different exceptions. Some researchers for example [5] distinguish an exception from its effect on causing failure, which means there are exceptions that may not lead to subsequent service failures. In this sense, a fault is active when it causes failure, otherwise it is dormant. For non-active faults, there are no manifestations or any signs of a mal-functioning, which makes it extremely difficult to diagnose or handle. From another perspective, if the overall service seems to be functioning correctly with the dormant fault, then the fault is not on a priority list to be handled, because it is not threatening any performance. In fact that is the key concept for fault-tolerance, to continue providing a core subset of services with the presence of faults. For those reasons, only active faults are considered in this research. Error is used interchangeably with exception here.

There are different granularities for Fault-tolerance [50]:

1. Full fault-tolerance, where the system continues to operate without significant loss of functionality or performance even in the presence of faults.
2. Graceful degradation, where the system continues to operate with some loss of functionality or performance.
3. Fail-safe, where vital functions are preserved while others may fail.

The objective for the fault-tolerance model here is to achieve graceful degradation.

2.3.5.1 Exception Detections and Fault Diagnosis with Context

Exceptions in computing systems can be viewed as abnormal events, which naturally require the definition of normal events to detect discrepancies. This is also the root for model based exception detection, which compares differences between actual event occurrence and normal event model. The modelling of events however, is a non trivial task especially in a dynamic and changing environment. Many events are transient and exceptions might not persist either. To accurately describe the system condition when exceptions occur, additional information that is relevant to the exception is crucial. The connection and interlink between components and processes have two impacts, firstly exceptions can propagate through these connections to have a bigger impact, and secondly it is difficult to isolate affected areas. The dependencies of components and processes need to be taken into account in the diagnosis process.

2.4 Data Interoperability Architectures

A physical data warehouse architecture is a classic example of a data integration architecture that integrates data into a joint repository based upon an analysis and harmonisation of their data storage schema. A virtual data warehouse is more of an example of a data interoperability architecture. In this section, several other major architectures for supporting distributed data interoperability and management are examined based upon distributed databases, client-server Web service models, the Grid, Semantic Web and Multi-Agent systems.

2.4.1 Distributed Databases

A Distributed database describes the situation where databases in more than one location are accessible but they are accessed as if were local. This enables data stores to be managed locally, e.g., environment databases reside in the regions in which they collect their data, but they can be queried as a collective, globally. Distributed databases are similar to virtual data warehouses but differ in the way in which the distribution is managed. Generally, the design restriction for distributed databases is that the data schema of the individual data stores must be the same or easy to align so

that either horizontal fragments of data, different instances of the same data types, or vertical fragments of data, different data entities, can be partitioned for distribution and recombined. The syntax of SQL is modified slightly to support remote requests, remote transactions (multiple data requests), distributed transactions and distributed requests [121].

2.4.2 Client-server Web Service Models

The objective of the Client Server Model is to reduce processing costs by splitting processing between clients and the server that reside on different networked computer nodes, e.g., File server and RDBMS servers are examples of client- server systems. For simpler or thin clients the main processing is associated with getting the data in a useful format for presentation. In more complex or fat clients, some post-processing can also be performed such as filtering and deriving data. Generally, two nodes or two tier client-server systems are static closed systems in that the clients are designed to support access to a particular server.

In a more open client-server model, a third node or tier is used to operate a directory service that specifies the public interfaces of resource services. If the clients know the access and interface to the directory service, they can browse service descriptions before selecting particular services and servers to interact with. In a more open service environment, clients and servers exchange message structures defined in public specifications such W3C SOAP specifications that adhere to the W3C eXtensible Markup Language or XML specification [31]. XML provides a language for users to define their own data structures. Another W3C XML specification called WSDL or Web Service Description Language is used to define the service descriptions. The OASIS XML based UDDI [118], Universal Description, Discovery, and Integration protocol defines the interface to the directory service.

UDDI is a platform-independent, XML-based registry for businesses worldwide to list themselves on the Internet. UDDI supports the registration of service capabilities from providers but does not contain any experiences of service users in the framework. The service provider oriented metadata structure encourages a single interaction mode, where users search for providers – they pull provider capabilities.

As service providers contain no information from or about users, they are unable to push their services to them. UDDI is not a semantic model.

2.4.3 Data Grids

The term “Grid” refers to technologies and infrastructure that enable coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organisation and that have been standardised by the OGC, Open Grid Consortium. A data Grid system consists of a set of basic Grid protocols used for data movement, name resolution, authentication, authorisation, resource discovery, resource management, and the like. A data Grid provides transparency in how data-handling and processing capabilities are integrated to deliver data products to end-user applications, so that requests for such products are easily mapped into computation and/or data retrieval at multiple locations. The focus of the Grid software community is defining APIs at the Grid level to access databases. More recently the Grid community have based their architecture upon XML Web-service models to access and process data. For database integrators to take advantage of data Grids they need to install and operate complex Grid software such as that based upon the Globus toolkit. The focus of Data grids is on high-performance and reliable data movement and replicating data possibly with a view to processing different parts of the same data parts in parallel [3].

2.4.4 Semantic Web

Semantics or meaning of data can be defined more formally using an Ontology model. There exists a range of Ontology model representations. At one end there are light-weight Ontologies such as dictionaries that have a simple conceptualisation and simple data structures have parts such as values of terms that may not be machine-readable and machine-relatable to other terms, e.g., XML and RDF (Resource Description Framework [96]). Whilst at the other end, heavier-weight Ontologies exist that support more descriptive conceptualisation and more expressive constraints on terms and their inter-relationships including logical constraints [18]. Regardless of the properties of the specific ontology, heavier-weight ontologies generally include

classes or data types; taxonomic relations between classes; attributes of elements of classes; relations between elements of classes and instances of classes and properties. In addition, in heavier-weight ontologies, logical assertions can also be defined for the classes, e.g. the OWL Ontology Web Language [85].

The focus of much computer science research is to develop ever more expressive ontology expressions, e.g., adding support for temporal constraints and more expressive logical inferencing. However, for information retrieval researchers and developers, it is more important that an ontology representation is easy to maintain, so that it can be embedded with legacy information systems such as relational databases and interlinked and synchronised to legacy data – the use of the Ontology model is an enabler to enhance information retrieval.

2.4.5 Intelligent Information Agents and Multi-Agent Systems

Intelligent agents are software entities that support autonomous, deliberative, proactive and goal-directed behaviours and can socially interact with other intelligent agents to form a multi-agent system. Each agent is capable of independent (or autonomous) action, acting on behalf of an owner that may have different goals and motivations, to satisfy user goals (goal oriented), rather than constantly reacting to what they are told.

2.4.5.1 Individual Agent and Multi-Agent System Properties

Multi-agent systems consist of two sets of agent properties – the properties of individual agents and the properties of multiple agents interacting. Each individual agent is capable of independent (or autonomous) action and acts on behalf of an owner. Russel and Norvig [95] define different properties and types of individual agent based upon how their actions are selected. The simplest is reactive agents that operate according to predefined event-condition action rule loops. A second type is a logical or modal agent that uses a logical model of the world to take into account and reason about how its actions affect the world. The third type is a goal-based agent that uses its desired end or goal state to select actions in its current state that move it closer to the goal state. The fourth type is an utility-based agent that can decide

actions when there are multiple goals or with goals that are multi-valued. The fifth type of agent is a learning agent whose performance of its actions can improve based upon learning with experience. Many agents in practice are hybrid types of agents that cover several of these properties. In the MAS distributed IR applications developed later in this thesis, the focus is on a hybrid of reactive, model-based and goal-based. This is because sometimes entities in a distributed IR system just need to react to message queries and requests. Sometimes entities need to plan a sequence of actions to reach a goal state and sometimes agents need to reason about the information they hold and receive.

The debate over what exactly constitutes the properties of individual agents is multifaceted. Different schools of researchers emphasise different aspects in agents. Some strongly advocate the notion of mental states of an agent to serve as the basis for their reasoning and decision making. The Belief, Desire and Intention (BDI) model is proposed for this purpose. Other researchers, who find a mental structure for a piece of software farfetched, stress on the social aspects an agent complies with, leaving internal reasoning as private to agents themselves. Human beings are restrained from conducting anything unlawful through the set of social conventions, i.e. government regulations, moral standards, or even social decency. Agents too can be constrained to a set of rules, and only external behaviours need to be monitored and restrained. The social approach needs not assume a specific kind of agent internal structure, as long as their external behaviours conform to conventions. This proposal also greatly simplifies the reasoning process in agents, eliminating the intricate deduction on mental beliefs.

Support for the set of behaviours of properties such as autonomy and goal-directiveness but without using mentalistic type notions such as beliefs, desires and intentions to guide behaviours has been termed weak-intelligence whereas the addition of mentalistic notions to drive the individual behaviours and interaction has been termed agents with a strong notion of intelligence [126]. In this thesis the focus is on leveraging the benefits of the weak intelligent type of agents.

MAS emphasise the social abilities of agents, to exchange information at the simplest level, to coordinate their behaviours so that a common goal can be reached, and to cooperatively solve complicated problems.

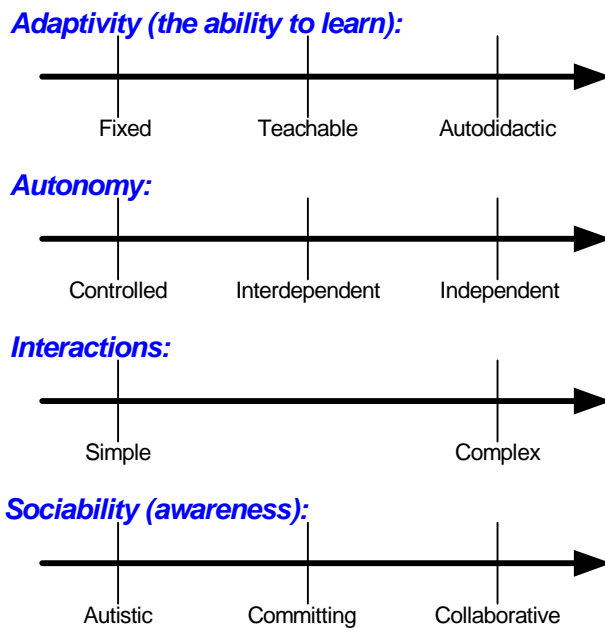


Figure 3 MAS properties: agents, derived from [55].

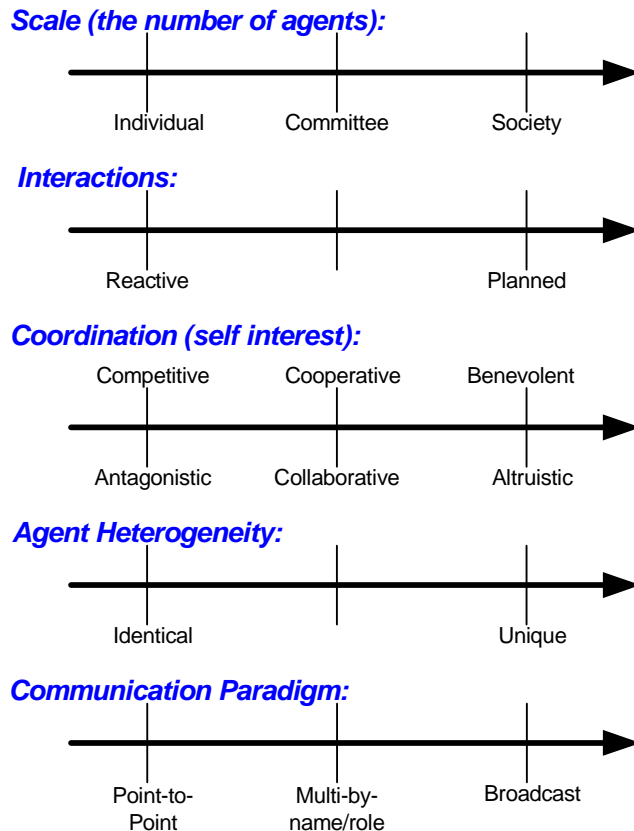


Figure 4 MAS properties: system, derived from [55].

To successfully interact, the individual agents will require the ability to cooperate, coordinate, and negotiate with each other, much as people do. Typically, this is supported by agents using an Agent Communication Language or ACL based upon speech acts [98]. Here the key features being investigated are their knowledge processing ability or reasoning, interaction and coordination mechanisms. According to [55], the properties of multi-agent systems can be discussed in terms of two groups of properties: adaptivity, autonomy, interactions and sociability, as shown in Figure 3 and on scale, interactions, coordination, and heterogeneity shown in Figure 4.

2.4.5.2 MAS based Information Systems

Specialised agents have been characterised called Intelligent Information Agents (IIA) for interfacing to data resources. This is defined as an autonomous, computational software entity that has access to one or more, heterogeneous and geographically distributed information sources, and which proactively acquires, mediates, and maintains relevant information on behalf of users or other agents, preferably just-in-time [69]. Generic requirements for IIA are grouped into three types, data acquisition and management (the data source front), information mediation, synthesis and filtering (the middleware), and intelligent user assistance including personalised data representation and data granularity (the end user front).

In distributed IR, agents can represent many different components of the system such as the data resources, the metadata directories, task coordination agents, monitor and management agents, user interface agents and so on. Based upon the general agent properties described in section 2.4.5.1 in relation to the problem requirements for distributed IR applications discussed in section 2.1.4, the specific types of MAS properties that seem useful for distributed IR applications are as proposed as follows:

- Fixed adaptivity: the intentional components, the structure and semantics of the core data resources are static, although the extensional components, in terms of new data instances being added and deleted over time, is dynamic.
- Interdependent and independent autonomy: many MAS are implemented using conventional distributed computing architectures. They are

interdependent on core middleware services such as centralised name and directory services, e.g., the chapter four framework. However they can be designed to be more independent and to utilise more of their own local services, e.g., the chapter five framework.

- Richer interactions: MAS can be designed to mimic conventional distributed systems mainly react using synchronous request-response interaction, e.g., to query database resources, and asynchronous notification interactions, e.g., to notify users of changes in data resources. However, MAS can also utilise a much richer set of interaction patterns for task and information sharing such as auctions, contract net and subscriptions.
- Rich sociability: this enables complex cross data resource queries from users to involve other agents as team players in a Virtual Organisation or VO in order to query directories. A team plan could be proposed on behalf of a user agent to locate the relevant resources, to then demultiplex a high-level query into individual queries that can be answered and then to multiplex the results.
- Benevolent coordination: agents can be used for cooperative problem sharing are designed to commit to help achieve common team or VO goals that are often initiated by user interface agents.
- Committee scale: most MAS applications involve the uses of tens to hundreds of agents
- Heterogeneous agents: in an open service environment services and data resources may be heterogeneous, this requires the use of heterogeneous interfaces to map local data and service types into virtual types to aid interoperability.
- From one-to-one to one-to-many interaction: much interaction is one-to-one, an agent users makes use of a directory service to select a service provider. However, in certain types of market-place interaction, information and task sharing requests can be sent to multiple recipients in order for service offers from multiple recipients to be evaluated by the initiator.

Agents can interface to non-agent components using (non-agent) service interfaces. This also promotes the reuse of best practice robust and mature non-agent service models such as relational database type data storage repositories and promotes leveraging MAS properties, e.g., agents could use joint plans with redundancy so that if one data source becomes unavailable, agents can re-plan to use another data source to achieve the same goal.

The reason to have multi-agent system is that collectively agents provide more powerful problem solving than individual agents because individuals are inevitably constrained by their limited scope and limited reasoning power. It is agreed that agent coordination is vital to the smooth running of a MAS, yet there is no consensus on a coordination mechanism. Some agents coordinate their behaviours by sharing interaction specifications on certain situations, while others negotiate to reach agreements. Defining and complying with social conventions is another approach to coordination. Both conversation protocols and social conventions are less flexible than negotiation in terms of facing unexpected circumstances. Yet the dynamic and open environment requires not only an efficient coordination mechanism, but also individual agent's ability to cope with uncertainty. Knowledge sharing helps to broaden agents view on the environment, and prepare for unforeseen situations. More specifically experiences in using services can be shared as well to improve their ability in handling changing situations.

2.4.5.3 Directories and Service Discovery

Many MAS are designed and implemented using client-server based object-oriented models. Most of the conventional directory facilities in MAS are centralise and only are service oriented – they contain metadata on the service capabilities. This introduces several deficiencies into the system. First of all, it is the performance bottleneck that faces serious challenge when the system scales up as many users may wish to concurrently consult the service directory. Secondly, it is a potential single point of failure, which in many cases leads to complete system halt. Thirdly it is too inefficient for users to query a central directory for metadata each time a data query has to be made. Because of the absolute importance of such a directory, many systems are designed in the way that the availability of the directory is a prerequisite

for the normal operation of the system. If the directory is not available when other agents are starting up, agents wait for the directory before any other actions are taken. This renders the system very fragile and inappropriate for a changing open environment.

The issue that directly relates to the structure of directory and organisation of metadata is service discovery. With a centralised directory, the discovery mechanism to a great extent relies on the client-server mode of interaction. However those deficiencies are highly undesirable, and in some cases catastrophic. To circumvent this, alternative service discovery mechanisms are proposed using more variable communication patterns and decentralised metadata model.

More specifically, two of the major indicators of middleware flexibility are the use of the directory and the use of other interaction related social metadata. Alternatives to conventional directory architecture and service discovery provide the means to alleviate the deficiencies introduced by a centralised metadata repository (see section 2.4.5.3). Another factor that improves the flexibility of the MAS in an open environment is to make use of service quality metadata to choose more suitable service providers. As to support for flexible query analysis that can handle high level complex queries such as the ones used by Decision Support System (DSS) or a policy maker, data warehouse is an option either physical or virtual.

2.4.5.4 Agent Frameworks for Information Integration based upon FIPA

The most widely used MAS framework for designing and implementing MAS systems and applications is based upon the FIPA model. The Foundation for Intelligent Physical Agents or FIPA, became an IEEE standards forum in 2005, was formed in 1996 to promote the uptake of software agents in businesses at large [38]. As FIPA progressed, the interest of the forum focused increasingly on software rather than on hardware agents despite its original intention. A second key focus was on specifying communication and interoperability between agents rather than on standardising specifications about how agents internally processed and reasoned about the information they received. That is, FIPA concentrated on standardising

‘external intelligence’ or rich interoperability rather than on ‘internal intelligence’ or reasoning.

The first set of 7 specifications included: an agent management specification that originally defined the concept of an agent platform; an agent communication language and some applications such as travel assistance, network management, audio-visual entertainment and personal assistance. In subsequent years, further modification to the existing standards and addition of new development continued. The FIPA specifications are now the most commonly used design for MAS application and have been deployed in a diverse set of domains.

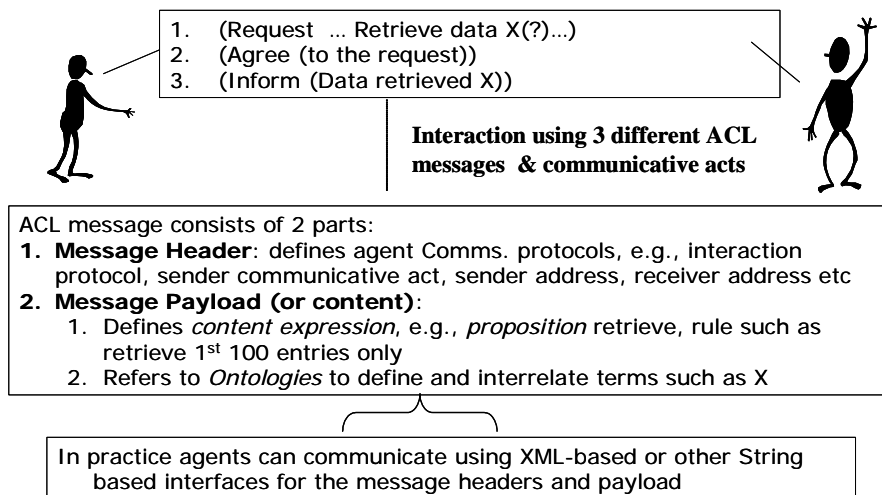


Figure 5 Overview of agent communication in which one agent requests information from another

The FIPA Agent Communication Language (ACL) specifications are based on speech act theory [98], which defines communication as an action. In practice, a FIPA ACL message instance consists of (Figure 5) message header with fields for Communicative Act or CA, Interaction Protocol, Content Language or CL, Domain Ontologies, and message body (see [40] for more detail).

As to using agent framework for data integration, one of the examples is the FIPA RDF Content Language specification, which uses RDF as a semantic protocol to encode content level semantics for information exchange. The FIPA RDF CL defines three: objects, propositions, and actions. The agent communication model adopted by FIPA follows the theory of speech-act, which fundamentally defines message

exchange as a means to invoke certain actions. There are mainly three kinds of information requested in an agent message: reference to an object, value of a proposition, and an action. As a resource description framework, objects required by the CL can naturally be represented as resources in RDF. And propositions are statements.

RDF is a very simple model, and it is limited as a choice for content language, as it is difficult to represent actions using RDF, and the lack of logic structure. Languages such as KIF (Knowledge Interchange Format) are designed to translate from one content language into another, which may seem ideal for the content language requirements. These knowledge translation languages however may be too complex for communications between agents and web services.

2.4.6 Architecture Choice: Hybrid MAS

An examination of the choice of architecture for a distributed information system is as follows. Each single type of architecture has its own limitations. After considering these, a case is made for using a hybrid architecture based upon MAS. Physical data warehouses are not considered to be a suitable architecture for a distributed environmental information system because of the lack of resources to create a large joint repository and because of the perceived loss of control and cost implications of data source owners to allow large parts of their data to be periodically distributed to others. The distributed database model is not considered to be a suitable option because environment databases are developed independently and autonomously, their data schema are often not similar and easy to align to support horizontal and vertical data recombination and to support distributed queries. Web services and the Grid support useful data exchange formats and services to support data interoperability and management but neither is considered to be a suitable option because it lacks support for dealing with data heterogeneities found in the environment information domain. The Semantic Web represents a good model for data exchange that can represent the data heterogeneities but it currently still lacks service models for actually mapping different semantic data and managing semantic data access. Ruling

out these choices leaves the multi-agent system architecture as the only remaining choice.

Multi-agent systems have the benefits of using rich service and management processes to support flexible and safe distributed information and task distribution and access but they also need to support specific models and services for semantic data interoperability, metadata management and directories, query orchestration and data aggregation and derived data analysis. Rather than designing a multi-agent system to support all of these, a hybrid system will be used to support existing models as follows:

- Data resource access, management: RDBMS
- Metadata management and directories; Web Services
- Heterogeneous Data interoperability; Semantic Web
- Query orchestration and data aggregation: MAS
- Derived Data Analysis: Virtual Data Warehouse
- Fault-tolerance for Query Management: MAS.

2.5 Summary

This chapter has looked into the issues of distributed information systems, in particular there are five main distributed query management concerns, namely metadata management and directory, heterogeneous data interoperability, query orchestration and data aggregation, derived data analysis, and fault-tolerance for query management. There are several choices for data interoperability architecture, and the pros and cons of each of them are discussed. Based on the fact that every single architecture choice has some limitations, a hybrid architecture is chosen combining MAS, Semantic Web and Web Services, Virtual data warehouse, and RDBMS. Related research work on distributed information systems based on agents or data warehouses to address data interoperability is surveyed in the next chapter.

3 Literature Survey

3.1 Scope of the Survey

Based on chapter two, the focus of this survey is on MAS approaches with extensions to support distributed information retrieval. Key projects and systems from both these approaches are surveyed, analysed and compared. The survey has three main threads. The first is the information integration systems that use semantic models to support flexible query analysis and processing. Two main approaches dominate this field, namely the agent or MAS approach and relational data schema based approaches that use semantics. The second thread concerns service discovery and directory services and mechanisms for MAS. Thirdly the agent-based fault-tolerance mechanisms for open services to taking into account the dynamic and uncertain nature of an open environment. Note however, these three threads may overlap, for example some MAS may support two or three of these, e.g., InfoSleuth [56].

3.2 Semantic Distributed Information Retrieval

There are two approaches to support semantic distributed information retrieval. Firstly, MAS can be used as the basic framework as they inherently support a semantics for interaction and support a semantics of action. Secondly, semantics can be added to relational database systems to assist more flexible information retrieval. Each of these is discussed in turn.

3.2.1 Multi-Agent Systems

MAS, in theory, inherently support a powerful model for semantic based distributed IR. MAS are organised as a collection of autonomous agents, e.g., representing the autonomous information resources under the control of individual owners. MAS can utilise a rich set of interaction protocols to share information and to share tasks. This can be used to share a query (task) to several data resources and to gather and correlate the resultant data. One of the most common types of individual agent

design is the model or logic based agent [95]. This can use logic based semantics to verify the truth of a query and to reason about the semantics of its associated terms and to infer new relationships and derive new terms to generate a logical result of queries.

Some of the key MAS and projects that have applied to support semantic based distributed IR include InfoSleuth, RETSINA, MACRON, SAIRE, BIG Agent, and ProFusion. This is analysed in more detail below.

3.2.1.1 InfoSleuth

InfoSleuth [80] is designed as an agent-based system for information discovery and retrieval in a dynamic open environment. InfoSleuth agents provide a number of complex query services that resolve ontology-based queries over dynamically changing, distributed, heterogeneous resources. These include distributed query processing, location-independent single-resource updates, events and information monitoring, statistical or inferential data analysis, and trend discovery in complex event streams.

Distributed query orchestration is designed using five types of agents: persistent user agents, temporal reasoning agents, information analysis agents, information extraction agents, and semantic brokering agents. Because the specific division of tasks within the agent community, patterns of tasks were defined using plans and a set of task plan templates were devised to be populated with parameters from query instances, and translated into executable agent interaction patterns [87]. Four primary classes of tasks are deployed to achieve four kinds of information-gathering goals: one-shot queries, periodic subscriptions, event-driven subscriptions, and temporal-analysis subscriptions. The InfoSleuth task sharing framework is in effect a rule-based the Hierarchical Task Network approach [87]. It is seen from the agent interaction framework that queries are routed according to their type, and agent collaboration is choreographed with specific plans. In other words the coordination mechanism is implicitly defined in the plan templates and is specified at the design stage, rather than relying on dynamic generation from agent negotiation at run time.

Analysis of derived data for decision support system, for example data aggregation across multiple dimensions, is not supported.

InfoSleuth utilises a peer-to-peer multi-broker design that involves both syntactic and semantic based match-making for service discovery. Syntax based matching covers agent naming and location services, communication services (using OMG IDL) and information access (using SQL, LDL). Semantic matching is based upon agent capabilities, agent content and agent properties. Brokers are federated in order to advertise and receive advertisements from each other, so that metadata is synchronised. A multi-brokering model is more robust than a single broker architecture, simply because of its redundancy but this can make it more difficult to manage. In addition, there is an overhead in message exchange associated with synchronising different brokers' copies of the metadata and maintaining their consistency. Semantic match-making of the capabilities of the service providers was claimed to be novel for a multi-broker architecture. But the capabilities are provided by the agents themselves and are effectively a claim of what they can do. This means there is no guarantee that the services actually work as advertised. Searches for services based on this knowledge are thus inherently limited because of the lack of mechanisms to verify these claims.

There is little published evidence concerning how exceptions are handled. There is an agent defined within the InfoSleuth system called deviation detection agent, which monitors streams of data for instances that are beyond some threshold. It is a method to check data accuracy. But this is just one specific kind of error handling out of many. It is not clear from the published papers whether data value deviation or other types of exceptions are taken into consideration during the design of the plan templates.

Service discovery in InfoSleuth is achieved using a multi-broker scheme. However a brokering scheme differs from a directory service. Brokering focuses on match-making requests to service providers, and the actual structure of service descriptions is less important. Only service capabilities are considered in the match-making process together with some basic syntactical information. The service browsing is

provider-oriented. It works based on users initiating search requests, i.e. a mode that users seek providers.

3.2.1.2 RETSINA

RETSINA, an acronym for Reusable Environment for Task-Structured Intelligent Network Agents, is a multi-agent architecture developed at the Intelligent Software Agents Lab in Carnegie Mellon University. It was designed as a Multi-Agent System to promote peer interaction between agents and to support coordination that emerged from the relations between them. The MAS infrastructure architecture had been differentiated from the agent architecture, because the infrastructure was believed to be domain independent. Some features such as the communication infrastructure, ACL infrastructure, name to location mapping and capability to agent mapping [112], are regarded as generic MAS components. However, the coordination schema and social norms are defined as application specific. The agent architecture uses four basic types of agents: interface agents, task agents, information agents, and middle agents. Each RETSINA agent has four reusable modules for communication, planning, scheduling, and monitoring the execution of tasks and requests from other agents.

The coordination scheme was achieved by a teamwork model, in which each agent committed itself to act as a member of a team and executed the team plan to achieve a goal. A RETSINA agent's knowledge and reasoning capabilities were defined in a declarative way, in terms of assertions, rules, constraints, and task descriptions. The knowledge is implemented as data and stored in the agent's knowledge base and Task Schema library [11].

In terms of information tasks, there were three types of information seeking goals an information agent received: a one-shot query, periodic query to the database, and monitor a database for an update. Adaptivity is a key property of RETSINA agents. It is partitioned into the following dimensions, the planning adaptation, execution adaptation, and the organisational adaptation [22]. As a result, robustness is achieved through adaptive behaviours by avoiding catastrophic events in a highly dynamic and unpredictable environment.

Because the design of the RETSINA system is not dedicated to information retrieval applications, the functions are not specifically tailored to the needs of information systems. Information interoperability that involves heterogeneous data sources has not been considered an issue in the design and implementation of the system. The main focus of RETSINA is its application independent agent infrastructure structure, and a capability description language called LARKS (Language for Advertisement and Request for Knowledge Sharing). For the reasons listed above, it is difficult to assess RETSINA in terms of a purposely built agent-based information system. Firstly, it is not clear from literature that RETSINA had specified a service discovery scheme to manage the distributed expertises in the system. Secondly because of the lack of support for data heterogeneity, RETSINA would be a weak framework to support heterogeneous data interoperability. However the adaptive properties of RETSINA agents are useful in coping with the open environment, although there might be large overhead associated with real-time negotiation to achieve the same goal.

3.2.1.3 MACRON

MACRON [21], Multi-agent Architecture for Cooperative Retrieval ONLINE, is an MAS model of Cooperative Information Gathering (CIG) designed specifically for complex information gathering environments. MACRON adopts the view of functional groups and query-answering units, which are analogous to the way organisations are divided functionally. Its DECAF (Distributed Environment Centered Agent Framework) architecture is based upon that used to test the GPGP (Generalised Partial Global Planning) coordination approach described in [20]. In the DECAF architecture, each agent has a task structure database, containing the agent's current view of its tasks and their relation to other agent's tasks, and is based on the TAEMS (Task Analysis, Environment Modelling, and Simulation) structure. In TAEMS, tasks and the hard and soft relationships between tasks are modelled for the purpose of coordinating and scheduling agent actions. The planning, coordination modules and local scheduler all make use of the task structure. The planner initiates specific task structures. The coordination module then suggests new meta-level tasks with possible scheduling constraints by evaluating the current task structures. The local scheduler takes as input the current task structure, local commitments (actions

for itself), and non-local commitments (actions for other agents) and produces several possible schedules. A “Design-To-Time” (DTT) heuristic scheduler is used, which generates a schedule that maximises the performance criteria while meeting the deadline constraints. A final schedule is chosen by the decision-maker based on the agent’s current, dynamically changing view of its performance criteria. For the CIG domain, the main performance criteria of interest are: speed, completeness of the solution, the certainty of the solution, network resource usage, and monetary costs. Lastly an execution monitor oversees the executions of actions from the current schedule, to anticipate problems and slipped deadlines.

MACRON does not address the issues of data heterogeneity support and directory services and service discovery support. MACRON does not address fault-tolerance either.

3.2.1.4 SAIRE, BIG Agent, and ProFusion

There are several additional projects, SAIRE, BIG Agent, and ProFusion, but these do not focus on heterogeneous data interoperability using MAS designs.

The Scalable Agent-based Information Retrieval Engine or SAIRE [81] is a multi-agent search engine employing intelligent software agents, natural language understanding, and conceptual search techniques to support public access to Earth and Space Science data over the Internet. In the attempt to relax restrictions on what the users can ask, SAIRE uses a natural language system that analyses user inputs as natural language phrases, and there is a process for the system to learn new words with the help of users defining their meanings. There is a coordination agent with the responsibilities of directory and broker. It is also in control of the communication between Agent Managers. There is no explicit agent reasoning process associated with coordination. SAIRE does not address service discovery.

The Bounded Information Gathering or BIG [73] is designed to work with specific types of data sources i.e., web documents. BIG is an information-gathering agent that processes Web documents to create product models and recommend purchases based on user selection criteria. The core of BIG is a resource bounded search, which also

makes use of the TAEMS structure with DTC (Design-To-Criteria) scheduler. BIG agents are able to modify their information retrieval tasks according to resource constraints. The MAS extension to ProFusion [32] aims to provide intelligent adaptive Web search that not only locates relevant information sources, but also adapts to changes in the dynamic Web environment. The key to this is the agent adaptation algorithm specifically tuned to its search engine characteristics such as performance, response time, and new result formats. Clearly both BIG and ProFusion MAS focus on enhancing the individual agent's ability to cope with complex environments, in terms of agent's reasoning or adapting ability respectively. Coordination between agents is not an issue, neither is the need to orchestrate query processing and to support service discovery.

3.2.2 Semantic based RDBMS

Both schematic and semantic data warehouse approaches have been introduced in section 2.2.3. The semantic approach is more flexible and extensible in terms of accommodating new data sources and adapting to new application requirements. Here only data warehouse based information integration systems incorporating the semantic models are surveyed.

3.2.2.1 InfoMaster Virtual Data Warehouse

Infomaster [45] is an information integration system that provides integrated access to multiple distributed heterogeneous information sources on the Internet. It creates a virtual data warehouse. To end users it offers the same functionalities as a data warehouse, without physically combining copies of data from different sources. The core of the Infomaster is a facilitator that locates the data source, designs a strategy for answering the query, and performs translation from raw data to required formats. Data sources are wrapped to mask DB-specific differences such as the different DBMS used. Rules and constraints are used to describe information sources and translations among these sources, which are stored in a knowledge base. The facilitator is naturally the centre of the system where queries are translated into DB specific ones and directed, and results collected and harmonised. An example is given using a virtual database with a reference schema that integrates four car

databases, and is able to answer queries on the virtual database using various data harmonisation rules, such as the one that defines the conversion between Japanese and US currencies. The rules are dedicated to translations between the reference schema and connected data sources. As to the representation of the knowledge base, it has not been clearly stated whether the semantic representation is explicit or not. There is a lack of support for high level data analysis functions, for example there is no analysing on any trends in car sales. The flexibility of the framework is difficult to comment on, because information on how the rules are produced and maintained is not available.

3.2.2.2 Toivonen Semantic base OLAP Cube

The use of semantic models for the creation of OLAP cube in data warehouses has been investigated in [117] where semantic languages are used to describe data sources to assist the user in designing a suitable OLAP schema to perform the Extraction, Transformation and Loading (ETL) processes. More specifically, RDF is used to describe the basic structure of the OLAP cube and OWL is used to describe the semantic model of world trade data. The main benefit of having semantic descriptions of data sources is that these facilitate the interoperability across applications. Also some processing can be partially automated with the help of semantic models. However it only adds semantic descriptions to the OLAP cube structure, the actual data analysis process has not been changed accordingly. Also significant extensions of the semantic models are needed, such as the inclusion of dimension definition and dimension hierarchy because OLAP is most commonly used together with a multi-dimensional data structure, in order to demonstrate the further benefits of the methodology.

3.2.2.3 Vdovjak RDF-based Semantic Integration

Vdovjak [120] proposes an architecture to provide semantically unified interface for querying heterogeneous information sources that combines the use of on-demand retrieval and semantic metadata. There are five layers in the architecture, namely source layer, XML instance layer, XML to RDF layer, inference and mediating layer, and application layer. It does not aim to create a single domain model that merges all

possible sources to provide a cumulative view of all attributes. It is noted that one global conceptual model cannot easily accommodate all possible application requirements. However the criteria to decide what goes into the domain model have not been clarified. The arguments for choosing RDF as the modelling language for the conceptual model are to some extent controversial, for instance whether RDF is sufficiently expressive to capture semantic relations of the data sources. It is true that at the time of the publication, semantic web languages were not mature. The RDF approach does have some benefits such as an easier integration with other XML-based structures.

3.3 Directory Service and Service Discovery Mechanisms

There are several pertinent issues concerning directory services: architecture, matching algorithm and the fairness of the decision process. A centralised directory can act as a bottleneck and single point of failure as mentioned previously. The distributed structure is able to tackle some of these deficiencies at the expense of more complex management. Hybrid directories with a central repository and distributed local copies of partial metadata also exist, which seek to combine advantages of both approaches.

The matching algorithm of a directory refers to the process of identifying the right services with the right capabilities as required by the data query. There are several related issues, for example can the matching algorithm handle non-exact matches, or does it return null if no exact match is found? Another complication emerges from the requirements for different data granularities. Different groups of users have varying degrees of expertise in the domain, and different interests. Therefore data needs to be aggregated, harmonised, analysed, and presented at different levels of granularities to best suit a range of users' needs.

When multiple services' capabilities match a service request, the directory's decision process has to choose among them. There are several options to report matches such as always return the same first match but this depends on the ordering of matched services names or capabilities. This can penalise services that are usually ordered last. Another option is to randomly return a match, this seems fairer to all providers. Yet another option is to return all possible matches and let the user decide.

Directories are queried based upon the stored service capabilities. Many directories also contain information on how to locate and invoke a service. This information is usually supplied by services themselves. So the directories assume that service providers are benevolent. Even if service providers act benevolently, they may offer poor quality or performance services because either they do not validate their quality or factors outside their control may lessen their service quality. Service quality is most usefully determined from the experiences of the users.

The FIPA approach to directories is to establish platform-wide repository of agents, and each agent registers itself with the Directory Facilitator at start up [90]. It adopts the client-server mode of interaction. The deficiency of such a structure has been discussed in section 2.4.5.3. Service descriptions cannot be (hierarchically) organised – this makes it inefficient to search as the number of entries increases. The search function provided by DF is quite simple and supports only exact matching. Inter-platform service discovery lacks a mechanism to federate DFs from different platforms. The FIPA DF model is ultimately a central model, and most probably the whole platform would fail to function if DF stops working.

Java Agent DEvelopment Framework or JADE [57] is a FIPA-compliant Multi-Agent platform. JADE has implemented FIPA's DF as one of the platform component, to maintain the list of agents with their service descriptions. The search interface of DF provides basic match-making functions. It is limited, because the service description structure does not support hierarchy. For example a service description has several services and several protocols which are treated as separate lists, without links to connect the services to their corresponding protocols.

It is clear that the incompetence of the available standard specifications leads to the lack of an efficient and dynamic structure. Both FIPA and JADE directory structure is provider-oriented, i.e. metadata describes provider information and supports client-searches-for-service mode of interaction.

3.4 Fault-tolerance for Open Services

3.4.1 Exception Handling in Distributed Artificial Intelligence

Exception Handling in DAI is a non-trivial task and many researches have attempted to address the problem with various models and methodologies. Two of the main evaluation criteria on these models presented in [89] concern how distributed their approach is, and how proactive and autonomous the entities in the system are. The distributiveness of an Exception Handling (EH) approach is closely linked to whether EH functions are built into individual agents, or whether or not they form a unique EH structure on their own rights. The former built-in EH function in agents is an agent-centric approach, while the latter is a system-centric model.

3.4.1.1 Klein Exception Handling Service

Klein [66] argued that agent-centric approach is problematic, because it is not only costly in terms of agent development, but also the built-in EH functions have to be carefully coordinated with the agent's normative behaviours which further complicates the agent. In addition, individual agents have only "localised" view on an error instance, hence lack a more systemic knowledge on the context sensitive error event that is required by the exception handling process.

The major objection against a system-centric approach is that it imposes hierarchy into a system where components are supposed to be equal and autonomous. Yet it can be argued that hierarchy is necessary for the management of a complex system, with some entities taking more responsibilities than others. It is analogous to the way human society operates. Here published work is reviewed and analysed in the following aspects: is the reasoning process distributed or centralised, is exception context been taken into account in the handling process, and is metadata being used to assist EH.

The domain independent exception handling service proposed by Klein [66] is motivated by separating out EH functions from agent functional development. There are three main novelties, the first is that the EH service applies a knowledge base of

generic exception detection, diagnosis and resolution expertise which can be applied to a wide range of domains. The second is that agents understand a standard language that provides at least a basic level of self-awareness and self-modifiability. The third is the implementation of the service as a set of standard agents that can be “plugged” in to any agent system whose agents support the mentioned language interface. Exception detection is assisted by the process taxonomy developed in the context of the MIT Process Handbook [66], containing substantial coordination mechanisms and problem solving processes. Normative “correct” behaviours are mapped against at run time to identify any discrepancy, i.e. detecting exceptions. Decision trees have been used for both the exception diagnosis (selecting the most probably cause of exception) and resolution (identifying the most suitable handling strategy) processes. Strategies are represented as executable script templates whose actions are described using the action language. An example of the agent death exception was specifically addressed using the EH service (see [68]).

Generally speaking, Klein’s EH service framework is a comprehensive work, and it combines a few research threads in MIT, such as the MIT Process Handbook and pattern matching tools. The separation of control structure from agent normal functions is beneficial to both the agent development and refinement of the exception handling technique. However it assumes a benevolent agent environment, and also it imposes certain level of control over problem solving agents. Although it claims that the service is domain independent and can be applied to different applications, the claim has not been verified in the paper. As to the above evaluation criteria, Klein’s EH service is a centralised reasoning model. Also the communication between agents and the EH service to collect exception information and distribute instructions uses a complete different language to the one used between agents, which can be seen as a considerable overhead. The use of metadata has not been described in the paper.

3.4.1.2 Haegg Sentinel Approach

Haegg [50] proposes special agents called sentinels to guard certain functionality and to protect from undesired states. Sentinels form a control structure to the MAS, and through the semantic addressing scheme they can monitor communication, build models of other agents, and intervene according to given guidelines. The idea of

separating EH functions from “normal” agents facilitates the agent development of core functions. The control structures can be added later on. Sentinels are designed to monitor specific functions, with checkpoints, where an agent’s behaviour and internal states are evaluated, and exceptions are alerted when found. Sentinels communicate with each other using the semantic addressing scheme, which enables communication without knowing the actual physical addresses of the other parties. Sentinels have plans that define handling strategies for exceptions under specific circumstances.

3.4.1.3 Kumar Teamwork Model

Kumar [70] argued that the teamwork theory can be used to create a robust brokered architecture that will recover a multi-agent system from broker failure without incurring undue overheads. It was also claimed that traditional exception handling techniques for distributed system cannot be directly applied to MAS without extensively modifying the underlying agent infrastructure. Rules have been defined to regulate how the team works, and a recovery scheme is devised in very specific steps as to what member of a team should react to what situations. In this sense, the exception handling strategies are predefined and not flexible in the way that unforeseen situations cannot be dealt with using the strategy. Middle agent failure is only one type of failure a MAS can encounter. This shows that the exception handling sub-routine is failure specific, and it needs to be modified if other types of failures are to be handled. There is no mention of use of a specific metadata model.

3.4.1.4 SaGE, AERO, and MoCA

The SaGE [104] model is an exception handling system dedicated to multi-agent systems that considers the following issues: the preservation of the agent paradigm, the concurrency and asynchronous communication means, and the social organisation of agents. More specifically, key exception handling concepts are defined and explained including such as exception, program unit, handler, handler scope, and execution context, and some others. Handler scope refers to the set of program units the exceptions of that can be treated by a given handler. An execution context is then created and associated with the called unit, to help identify the right

handler. This is particularly important to effectively handle exceptions. Specific handlers are designed respectively for agents, services and roles. Handlers are first searched locally and then within the enclosing context. A concerted exception mechanism is also designed, to allow acute criticality evaluation. In particular, the ideas of handler scopes and exception context are interesting, in that they both contribute to narrowing down the affected area and pinpointing the actual cause.

AERO [16], Agent for Exception Recovery Outsourcing, is designed to handle exceptions in mobile agent systems. Basic exception handlers are represented as mobile agents that are dispatched to the platforms where exceptions occur. The outsourcing approach is neither “agent-centric” nor “system-centric”, therefore it addresses some of their limitations, such as putting too much computational complexity into individuals for the agent-centric approach, and require too much communication overhead and coordination for the system-centric one. The paper considers the design decisions of how to organise the exception handling service into particular agents with particular roles, and when and how some of these agents will migrate to an agent system to provide exception handling. Experimental results in handling the timeout dysfunction exception show AERO significantly reduces the cost of exception handling than without AERO.

The exception handling strategies in some prototype context-aware collaborative applications built with the Mobile Collaboration Architecture or MoCA are presented in [19]. Context is defined as any information that can be used to characterise the situation or an entity. A system is context-aware if it uses context to provide relevant information and/or services to the user [19]. Issues such as management of exceptional contexts, context-sensitive error propagation, and execution of context-aware exception handlers are discussed. Exceptional events are not only associated with contexts, but also linked with specific scopes. Three types of scopes are defined, namely the device(s), regions, and groups. The contextual exception is systematically propagated to broader scopes until the appropriate handlers are found. Exception handlers take into account the specific contexts. The exception contexts and exception scopes seem to be fixed, with the specifications of the exceptions themselves. The details on the modelling and representations of these contexts and scopes are not disclosed.

Some other existing work includes using replicates to handle exceptions in MAS [33], an event based approach of monitoring visualising system operation to help human experts to diagnose an exception [128], a flexible EH model that employs individual agent's exception handling mechanism for some social exceptions and uses the sentinel agents for others [99]. The replicate approach has to tackle the tricky problem of synchronising between states of the replicate and the original. Agents' states are so dynamic and unstable that makes the constant synchronisation nearly impossible. In addition, an agent's state is internal to the agent itself, therefore accessing and duplicating an internal property is to some extent against the principle of autonomy. The event-based mechanism monitors system operation and compares observations to models of corrective behaviours to detect any anomalies. This is not a fully automated exception handling approach, as it only provides a solution to fault detection. Fault isolation is left to human experts and fault clearance is not covered in the model. Finally the flexible model presented in [128] is basically a minor extension to Klein's framework.

3.4.2 Exception Handling in Autonomic Computing

Autonomia [27] is a proof-of-concept prototype system that can support the self-configuring, self-deploying and self-healing of any networked application. Three types of faults are considered, namely node fault, component fault and agent fault, with three corresponding handlers. The fault recovery procedure is based mainly on the check pointing and migrating mechanism that is one of the conventional fault recovery methods.

There are several published research papers covering the issue of fault-tolerance in autonomic computing, such as [107]. However they focus on presenting the idea that autonomic computing as a relatively new paradigm that could be a promising approach to provide high level dependability.

Several challenges for exception handling in autonomic computing environment have been identified in [116]. The paper also presents some general ideas on how to design solutions and some issues worth taking consideration into. However it is not

the aim of the paper to define specific solutions, but outlining the problems and opportunities.

It is part of the aim of autonomic computing to design self-defining, self-protecting, self-optimising, self-healing, self-configuring, contextually aware, open and anticipatory components so that a higher level of automation can be achieved. It assumes that fault handling is part of the capabilities or prerequisites of individual components so that they can self-heal in case of exception. System-wide exception handling through co-operations is therefore not considered a key issue. As to the realisation of individual's self-healing capability, there is little evidence to support this currently.

3.5 Discussion

This chapter has outlined some major projects and systems across the three main dimensions of interests: distributed query orchestration, Directory service specifications and service discovery mechanisms, and fault-tolerance for open services. Each of the projects has been described individually with their limitations. The cross-comparison between them is shown in Table 1.

Table 1 Summary of surveyed project limitations comparing to desirable features

Surveyed project	Distributed query orchestration	Directory structure and service discovery	Support for flexible data analysis	Fault-tolerance
InfoSleuth	Agents share tasks, and the orchestration is done using plan templates	Multi-broker matching making, no semantic matching	N/A	A deviation detection agent is used to check data integrity. Other faults are not handled.
RETSINA	Coordination is done using teamwork model	No explicit directory structure.	N/A	Agents are adaptive to avoid unexpected circumstances, but not trained to handle specific exceptions.
MACRON	Coordination is done with	No explicit directory	N/A	N/A

	task sharing model.	structure.		
InfoMaster	N/A	N/A	A virtual DW, no explicit semantic description of the data structure	N/A
Toivonen Semantic OLAP	N/A	N/A	Semantic model to describe OLAP structure	N/A
Vdovjak RDF-based integration	N/A	N/A	Use RDF to query heterogeneous data sources, but no support for data analysis	N/A
FIPA Directory Service	N/A	Centralised model, lacks inter-directory communication mechanism	N/A	N/A
JADE Directory Facilitator	N/A	Centralised model, flat service description structure, no federation between platform DFs	N/A	N/A
Klein's Exception Handling service	N/A	N/A	N/A	Central EH service, uses dedicated communication language for exception reports
Haegg's sentinel approach	N/A	N/A	N/A	Decentralised approach based on sentinels that communicate with each other to reason about exceptions
Kumar's teamwork model	N/A	N/A	N/A	Brokers form a team to tackle broker failure. It is exception-specific.

Moreover, distributed query orchestration requires coordination of multiple queries across distributed data sources and post-processing for result aggregation and data

harmonisation. The orchestration also involves translation of high level queries into low level ones with DB specific terms. None of the systems surveyed supports the distributed query orchestration very well.

In addition, current work on exception handling generally uses domain or failure specific approaches. Klein claims his model is domain independent but this is not verified. Few model has exhibited reusability of their EH strategy. None of the projects has used metadata to assist exception handling. The importance of exception context has not been fully realised, and only one mobile agent system [19] has used contexts in its exception handling model.

It can be concluded from the comparison shown in Table 1 that none of the surveyed system provides support to the four key desirable features very well.

3.6 Summary

To summarise, there are many agent-based projects for information integration, but they all have different focuses on the integration issue, and none has addressed decentralised metadata management, derived data analysis and query fault-tolerance all at the same time. Some research work also exist in using semantics with data warehouses to integrate heterogeneous data, but no project has combined agents with semantics in virtual data warehouses to provide a more flexible solution to high level data analysis problems.

4 EDEN-IW MAS to Integrate Distributed Databases

The main aim of the EU Environmental Data Enabled Network for Inland Water (EDEN-IW) project [28] is to make information about Inland Water (IW) quality in databases distributed across Europe, available to the public. EDEN-IW presents a challenging domain where distributed heterogeneous database sources need to interoperate to support more use-centred rather than provider-centred information retrieval, e.g., to simplify very complex information access so that is at a level of abstraction that the user can understand and access.

4.1 Inland Water Quality Data Domain

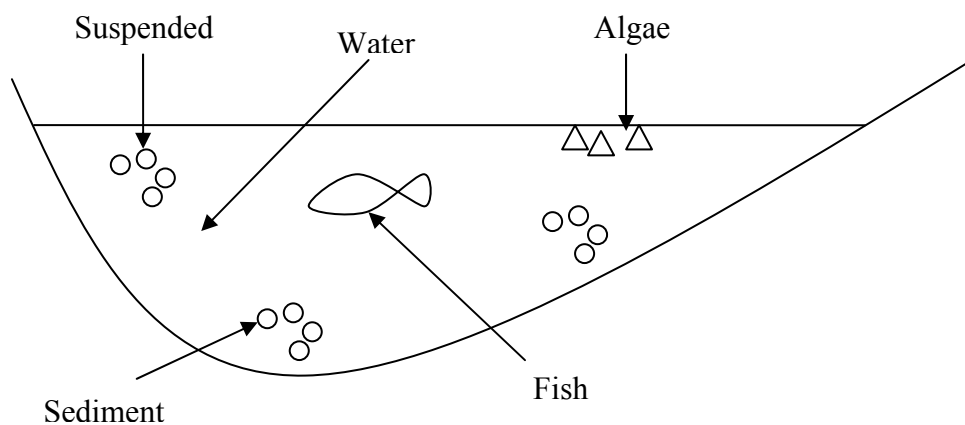


Figure 6 Key elements that affect water quality

Water quality data refers to chemical, biological, and physical measurements or observations of the characteristics of surface and ground waters, atmospheric deposition, potable water, treated effluents, and waste water and of the immediate environment in which the water exists [49]. Substances in the water body are the main subjects of measurements. The key components that affect the water quality are shown in Figure 6.

Inland water can be further divided into water from streams or rivers and lakes. There are many methods associated with how water qualities are defined, and how these quality data can be obtained from measurement. Any measurements are

inevitably associated with very specific factors such as the time and location of the event. Different countries may have vastly different geographies, which may directly or indirectly determine partly how the measurements are mapped to a relational data model. Although the key concepts are generally applicable in almost every water quality measurement system, how they are selected and modelled distinctively into a database can vary substantially from one data repository to another. That is one of the major reasons why data heterogeneity abounds in information systems.

4.2 Information Integration Infrastructure: MAS

The justification for the choice of a MAS architecture for distributed IR was analysed in chapter two and summarised in Section 2.4.6. The aim of this chapter is to describe the research and development of the basic MAS distributed IR system to support the requirements given in Section 2.1.4 because none of the surveyed systems analysed in chapter three could meet these requirements.

4.2.1 Architectural Overview of EDEN-IW

The overall architecture of the EDEN-IW prototype system is given in Figure 7. It consists of the following components: user interface that resides on the user Web portal, core MAS system consisting of user, task, directory and resource agents, IW domain ontology and ontology services, and the IW DBs accessed through a data owner Web portal.

This represents a three-tier architecture in terms of a user access / presentation tier, data resource / service tier and the mediator tier including the task and directory agents. The mediator tier masks the complexity from both end user at the web browser side and the data source at DBMS side. A traditional two-tier approach, i.e. the client-server model, is not flexible enough for use in a heterogeneous open service environment as it requires to clients understand and statically bind to multiple heterogeneous low-level resource and service interfaces. Utilising the MAS in the mediation tier makes information retrieval more flexible by putting complex processing into the middleware, masking complexity from both the user and service or resource provision ends. For example, metadata about each data resources are

retrieved and stored in a repository and ready to be used, and still available in the case of DBMS failure. The core agent system that acts between the two portals functions acts as a glue to seamlessly provide data access services to a variety of users from a variety of data resources.

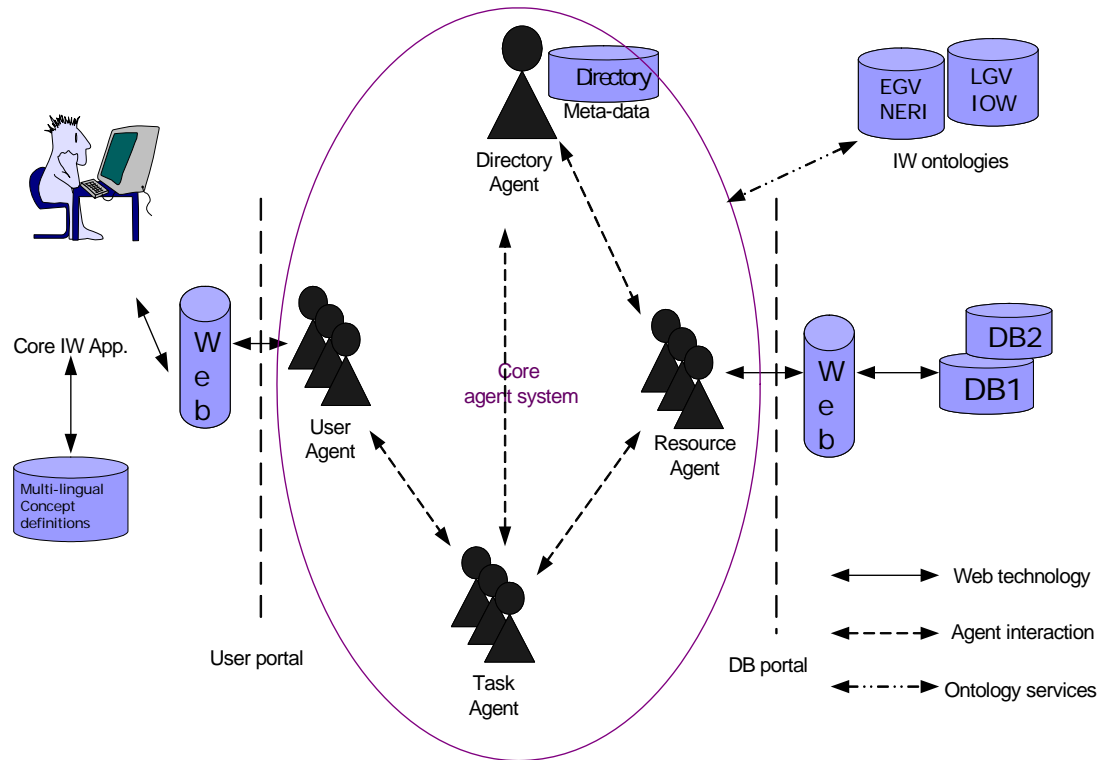


Figure 7 Architectural overview of the EDEN-IW prototype system

The architectural diagram shows the main messaging at the application level. From a technical point of view, the system is built using: the JADE agent platform [57] as the host for intelligent agents, HTTP as a common transport protocol, XML extensions such as RDF for data exchange and SQL for DBMS access. Agents which may be distributed within and across several JADE platforms, communicate with each other using FIPA Agent Communication Language (ACL) messages encoded in XML. At the user interface, a form interface is used to express data queries. This generates an RDF data structure for the data service action request. This is sent as part of the payload or body of an ACL message by the user agents to other agents in the system. The terms in the RDF service action structures are related to terms in the core Inland Water domain ontologies represented in DAML+OIL [108] that each agent accesses in order to interact within the domain. The Directory Agent or DA

contains both agent descriptions of other agents' services registered with it, including descriptions of data resources. These service descriptions are also represented in RDF, see section 4.2.4 for more detail. At the data resource end, data queries in the RDF format are translated by the relevant Resource Agent into SQL using local database terms. There is an access control mechanism enforced at the database owner site, where data access connections are required to have a valid user name and password to proceed.

The metadata models need to be initialised before the system can function, for example the Directory Agent needs to populate its metadata repository before any metadata queries can be answered to find relevant data sources. Therefore at the system start-up stage, Task Agents and Resource Agents register their agent descriptions with the Directory Agent, which also maintains a metadata repository of the data source summaries. Once the metadata repository is populated, the system is ready for data queries.

The creation, usage, and management of the domain semantic model is the focus of another PhD project (see [130]) that stemmed out from the EDEN-IW project. Detail of the domain ontologies development processes can be found in [130]. The main focus on this chapter is on the design of the MAS to support distributed query orchestration and results harmonisation.

4.2.2 MAS Design Methodology

The design methodology used for the design of the MAS model of the distributed IR application is a simplified AOSE, Agent Oriented Software Engineering methodology that has been described in part in [92]. This entails defining and developing four main types of model that are then implemented: the task plan model, the interaction model, the role model and the Ontology model. These four different models differentiate MAS engineering from conventional distributed object-oriented engineering:

- Role model: determine the set of activities (actions and interaction) an agent carries out as part of an organisation to its own goals that contribute to achieving its organisational goals. A role is a set of connected behaviours

(actions / processes), rights and obligations as conceptualised by organisational entities in a situation. It is mostly defined as an expected behaviour in a given individual organisational status and organisational position. Modelling the organisation using entities or roles is often a matter of preference and style. Roles support a more dynamic approach to organisations. Entities combine multiple roles and several entities can play the same role and entities can change roles at run time. The use of a role separates responsibility from identity. At a simple level the agent roles identify the main components in the system. Ferber [35] points out that organisation and interactions codetermine each other. Interactions are constrained by the role of the organisations and organisation roles are determined by the interactions they support. Thus, the role models are given in the interaction models.

- Task Plan model: organising the sequence of actions or tasks that the individual agent undertakes to achieve a goal, see the use of HTN described in chapter six, Section 6.3.1. In some cases some of these tasks will be delegated to other collaborative agents using one or more interaction protocols.
- Interaction model: agents select actions according to the type of agent and properties they support, see Section 2.4.5.1. In some cases, agent will evaluate that actions cannot be carried out by themselves but must be delegated and shared with other benevolent agents in the organisation. The interaction is determined by the role an entity holds in the organisation, see Figure 8.
- Ontology model: agents use Ontology based models to formally specify the semantics of information and tasks and to share these. A graphical model of an Ontology, for service quality, is given in Figure 18.

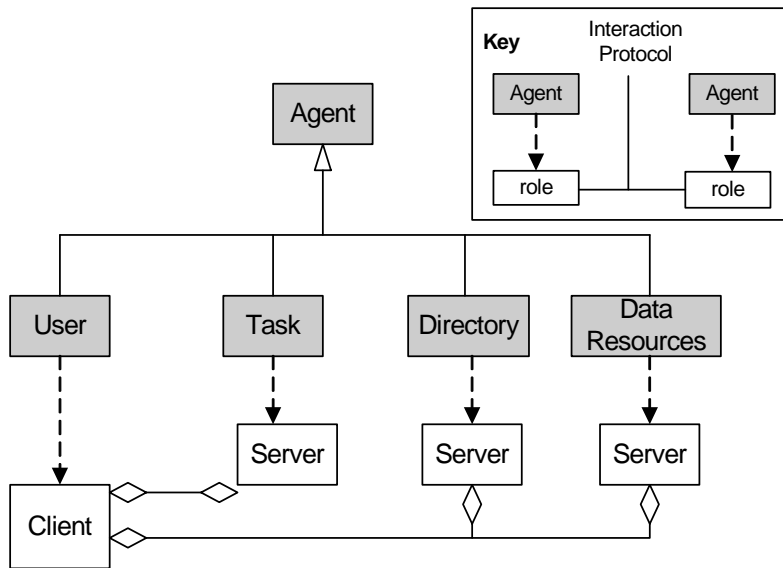


Figure 8 MAS role model for EDEN-IW system

4.2.3 MAS Component Design

Note that the MAS of the EDEN-IW system is a joint-contribution of participants in different work packages, and the author’s contributions mainly consist of the design and development of the Task Agent to support distributed query orchestration; the design of the Directory Agent as a metadata repository and the use of RDF service interfaces (see section 4.6.1 for more detail). The functions of the main components in the EDEN-IW MAS system are described in Table 2.

Table 2 System components and their functionalities

Name	Functionalities
IW Ontology model and services	Capture IW domain knowledge, provide mapping services that enable SQL formation in user queries using general terms.
Core agent system (User Agent, Directory Agent, Task Agent, and Resource Agent)	Provides query answering services, management of multiple user query sessions, connection to RDBMS
RDBMS	Provides data access via SQL

The agent functionalities and activities of the core agent system are given in Table 3.

Table 3 Agent functionalities & activities

Agent Name	Functionalities	Activities
User Agent (UA)	Receive user input and form data queries, forward queries to suitable service provides, and format results from services providers for the user	Send queries and process results. Receive updates on status of query processing.
Directory Agent (DA)	Manage repository of metadata and service descriptions; perform semantic matchmaking between queries and resources	Register / deregister agent description, send queries, process replies.
Task Agent (TA)	Query decomposition, task scheduling; interaction management; exception handling	Register with DF/DA, send queries, process replies, combine results, report query status
Resource Agent (RA)	Wrap data resources; query generation and execution; semantic mapping of concepts between user queries and IW ontology definitions	Register with DF/DA, SQL formation, send replies

As the User Agent and Resource Agent acting as portals to human users and data resources respectively, the Task Agent is responsible for managing interaction between them and providing a seamless query answering services. The MAS architecture together with the IW domain ontologies and metadata repository, make up an infrastructure for flexible task and result sharing. These are described in more detail as follows.

The EDEN-IW Directory Agent has been defined to supplement the basic FIPA DF Directory Agent, for example that has been implemented in JADE [57]. The DF maintains a list of agents with their service descriptions, but hierarchies are not

supported in service descriptions. The DF action of deregistration deletes the whole agent altogether. Also the DF support only pull-type of interaction not push-type of notification interaction. Hence, an enhanced Directory Agent or DA has been designed. This supports subscriptions to metadata updates from resource agents instead of having to explicitly pull information.

FIPA ACL communication can be interpreted as a multi-application protocol stack that builds on specifications of transport protocols, for example HTTP between web clients and servers. It then adds an Agent Communication Language (ACL) based upon communicative acts. It adds patterns for interaction sequences of individual messages between agents and adds explicit semantic specifications for sharing domain specific task and information sharing.

The latter is specified in the FIPA ACL model using a so called Content Language. This application sub-protocol supports semantic message exchange, i.e. it can specify semantic query constraints so that it can be correctly interpreted and processed. There are a range of possible semantic languages from weak semantic languages that focus on simple conceptual organisations without reasoning support, e.g., the W3C XML and RDF specifications to specifications of very expressive conceptual organisations with logical reasoning support, e.g., the W3C OWL. RDF is chosen as the representation of the semantics for task sharing because:

- It is based upon XML but is more expressive than XML because unlike XML that can only nest elements, RDF can name relationships between elements and hence chain relationships together.
- RDF is the basis for a suite of more expressive metadata languages such as RDF-S and OWL.
- RDF is also an experimental standard content language for the FIPA agent standards. A content language defines the application dependent part in the agent communication language, where action logic is described.

4.2.4 Metadata Structure

The EDEN-IW system metadata consists of three different parts: the metadata about the database schema, data summaries of the data stored in each of the main data

resources, stored in the directory agent and finally agent descriptions containing basic information about each agent communication and interaction.

4.2.4.1 Agent Descriptions

The Task Agent and each Resource Agent registers a description containing their name, address within a specific platform so the agent is contactable through this address, type, interaction protocols they support and the domain specific ontologies they use with the Directory Agent.

4.2.4.2 Data Summaries for Data Sources

For each data source, data are summarised according to three main IW domain concepts: location, time and water quality indicator. Based upon these summaries, specific data source can be retrieved given a certain determinand name or station name, within a given time-span.

4.2.4.3 EDEN-IW Core Domain Ontology

The EDEN-IW conceptual data model describes how the various concepts within the inland water domain are related. The EDEN-IW domain ontology has 3 main qualities as defined in EDEN project deliverable D11 [28]. It should be:

- Expressed in a computational form that describes the concepts and relationships found in the Inland Water Domain;
- Encompass heterogeneous data source models found in the Inland Water Domain;
- Extensible in accommodating new data sources without making too many changes to the way the model works.

The conceptual data model may be divided into different levels. At the top-most level an Observation is composed by the Value of a Determinand that is taken at a Station at a given Time. The water quality observation is also tightly associated with the medium and analytical fraction with which the measurement is taken. The conceptualisation between observation, time, station, determinand and value is

shown in Figure 9. More details on the IW domain ontology could be found in EDEN-IW project deliverable D18 [28].

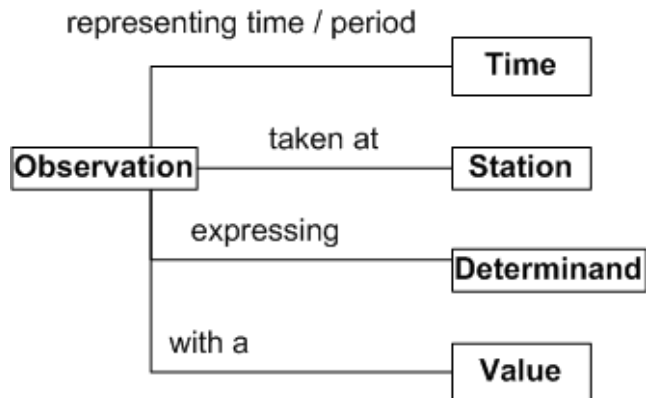


Figure 9 Overview of conceptual data model of an observation

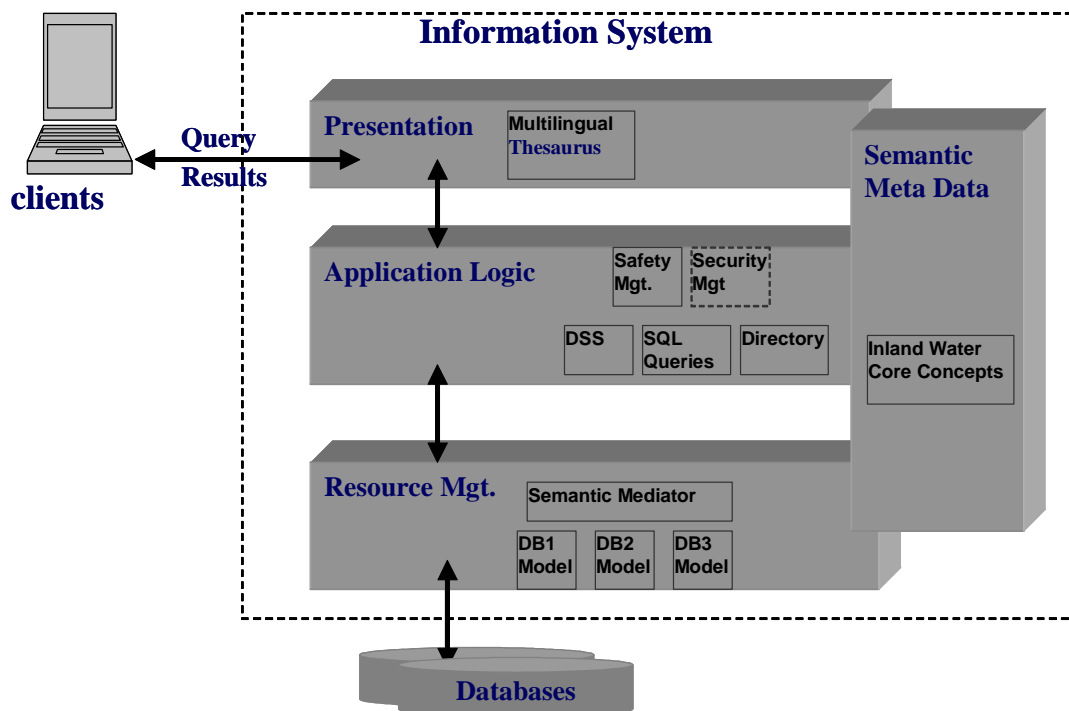


Figure 10 The EDEN-IW Semantic data model of core Inland Water Concepts interlinks the different parts of the distributed information system: Presentation logic or application processes, e.g., Decision Support System (DSS); Resource management processes such as Semantic Mediation and the local database interface models (DB1-3).

Figure 10 shows that there are often multiple ontology applications within a single (IW) Ontology domain, multiple heterogeneous databases with different schema in multiple RDBMSs and multiple user viewpoints of the data. It can be seen as an

instantiation of the generic information system schema shown in Figure 2 of chapter 2. The semantic management layer contains the semantic mediation processes with local mappings to data sources; the application logic layer handles distributed query processing, organisation of the directory service, support for DSS queries, and query fault-tolerance; the data presentation layer consists of a multi-lingual thesaurus; the semantic metadata model is made up of domain ontologies that capture key domain concepts and their relations. The semantic metadata model interplays with all three processing layers.

4.2.5 Agent Interactions

4.2.5.1 Query Interfaces

The queries supported in EDEN-IW system consist of:

1. Data retrieval queries:
 - GetParameter (UC1): which is the Observation Value(s) of Determinand X measured at Station Z between time period T1 and T2?
 - GetParameterLocation (UC9): what are the stations where Determinand X has been observed?
 - GetParameterLocationWithThreshold: what are the stations where Determinand X has been observed with a value that is greater than (less than or equal to) threshold T?
2. Meta data queries, for semantic match-making between query and data resources:
 - GetStationList: give me all the stations in your database.
 - GetDeterminandList: give me all the determinands in your database.
3. Service discovery query, for agents to locate other agents with specific expertise using the metadata directory:
 - GetResourceAgent: give the resource agent information in directory repository that has station X or parameter Y.
 - GetTaskAgent: give the information of an available task agent.
4. Agent registration request, for directory agent to manage metadata repository about agents available in the system:

- RegisterAgent: register agent X with the following information in directory repository.
- DeregisterAgent: delete the record of agent X with the following information in the directory repository.

4.2.5.2 Query Representation Layer: RDF

RDF was chosen as the semantic content language for task and information exchange within the EDEN-IW system because it was one of the content language recommendations from FIPA standardising body. In addition, RDF helps to bridge the communication gap between the use of agent for semantic coordination and exchange of messages. EDEN-IW does not require complex logic to reason about the information and tasks exchanged, hence RDF suffices.

According to the FIPA RDF Content Language specification (XC00011B see [41]), an action expresses an activity, carried out by an object. Three properties relate to an “action”: an act identifies the operative part of the actions; an actor identifies the entity responsible for the execution of the action; and an argument identifies an (optional) entity which can be used for the execution of the action.

This basic structure was extended in EDEN-IW, with more metadata tags and a corresponding part for query results as apposed to arguments. More specifically, a data retrieval type query has four parts: the header, the input, the query status, and the result. The header includes the actor and act tags, meaning the intended receiver for the query and basically the query type respectively. The input part is simply an aggregation of input values, such as determinand ID, station name, and date. In order to support query status, the done tag indicates that if the action has been successfully carried out, and the effect tag gives the information whether the query result is complete or not. The result part has a collection of retrieved data. Other types of messages also have similar fields, for example an agent registration message still has the header, argument (instead of query input), and action status, the only difference is that it does not have the result section, because an registration requires only an inform message or failure if error prevents the registration to complete.

Agent descriptions are described in terms of: agent name, agent address, agent type interaction protocol, and ontology supported. All task agents and resource agents register with the directory agent when they first start up. User Agents are not required to register with the Directory because searches are designed to be provider oriented - users search for providers. The Directory Agent keeps records of the agent descriptions until certain agent deregisters itself. The following message in Figure 11 is an example how TA registers itself with DA.

```

Header=actor(DA),
        act(RegisterAgent).
Input=AgentAddress(ta@jannylaptop:1099/JADE),
        AgentType(TaskAgent),
        AgentName(ta),
        AgentProtocol(FIPA-Request),
        AgentOntology(EDENGlobalOntology).
MessageStatus=done(false),
        effect(pending).

```

Figure 11 Summary of an agent registration message from a Task Agent to the Directory Agent

A use case 1 query (*what is the concentration of determinand X at station Y during time period T1 and T2*) is summarised as shown in Figure 12.

```

Header=actor(TA),
        act(GetParameter).
Input=DeterminandID(4),
        MediumID(*),
        AnalyticalFractionID(*),
        UnitID(*),
        StationName("POGNY"),
        StartDate(Year=1980, Month=1, Day=1),
        EndDate(Year=1990, Month=6, Day=9).
MessageStatus=done(false),
        effect(pending).
Result=DeterminandName(*),

```

ConcentrationValue(*), MediumName(*), AnalyticalFractionaName(*), UnitName(*), StationName(*), Date(*), DatabaseName(*).
--

Figure 12 Summary of RDF fields in a UC1 query

4.3 Distributed Query Orchestration

The query orchestration is mainly the responsibility of Task Agent, which is in control of: decomposing queries, locating data sources, distributing sub-queries, and assembling results. A directory service based on semantic metadata is used to facilitate service discovery.

4.3.1 Task & Result Sharing Mechanism

4.3.1.1 Main functionalities of TA

The Task Agent plays a key role in the query answering process especially when the system scales up. More specifically, it has to:

- Manage multiple interaction sessions between agents;
- Locate relevant resources by querying meta data repository;
- Pre-process queries i.e. translate queries into answerable ones for directory agent and resource agents. For instance a metadata query asking for all the available determinand in DBs needs to be translated into queries in DB specific terms for each data source;
- Schedule and plan interactions between agents and services;
- Post-process the results, i.e. results assembly and aggregation.

Apart from the query-processing functionalities, the TA also performs:

- Monitoring the query status, i.e. it keeps a record of the queries that have been processed and holds information about the process time, process duration and result size etc.
- Exception handling: this focuses on reporting errors to web page in user terms, e.g. when the results from the database are not sufficient, alternatives can be offered to users.

Take a particular user query as an example, breaking down the query answering process will give an insight into how those functionalities are realised. One of the most common user queries in EDEN-IW prototype system is “*what is the concentration of determinand X in station Y during time Z*”. Upon receiving the query, TA starts pre-processing, i.e. extracts key words from the query and formulates plans according to the key words. The plan generation process is discussed in detail in the next section. Here a goal refers to giving UA an answer, and a sub-task would be for TA to retrieve metadata from Directory Agent. After the pre-process on the incoming query, a task is decomposed into sub-tasks, and a plan managing the sub-tasks is generated. TA then maps the plan into executable actions, i.e. formatting a metadata query for Directory Agent. As there might be more than one conversation taking place at one time, TA is also making sure that interactions belong to the same conversation would not be confused with another conversation. This is the multiple interaction session management. When eventually results are returned from the data resource via resource agents, the TA may need to resemble them. That is if a query goes into more than one database for data retrieval, results from the DBs are aggregated in the TA. Result assembly is one of the functionalities in the query post-processing.

4.3.1.2 Scheduling Tasks and Query Post-Processing

Scheduling is somewhat similar to service orchestration, which refers to the ordering of individual steps in an overall composition process. The implementation is based on the JADE [38] platform. JADE agent platform provides message transporting infrastructure and agent actions that are modelled as behaviours. Agents interact with each other by exchanging messages. Naturally tasks are message oriented, so scheduling tasks in this case means scheduling agent interactions. From the TA’s

point of view, it needs to manage multiple interaction sessions with different types of agents, and different instances of the same type of agent. Conversation ID is used in each message to distinguish the interaction session.

The post-processing of queries now includes combining results from different data sources into a single reply, e.g., unit conversions on the data, i.e. if two DBs are using different units on the same determinand. When combining results from the two sources, post-processing is needed to convert data value from one unit to another, e.g., from mg/ml to mg/l.

Apart from managing multiple sessions of agent interactions and post-processing the data results, the TA also sends status message back to UA in order to keep track of how the query is being processed in the system. The TA pushes the status information to UA, i.e. whenever the query status is changed, it sends out the report to UA without the UA explicitly requesting it.

4.3.2 Service-Action Model

Agent interactions can be classified into four types (see section 4.2.5.1). The first two are data-specific interaction (interaction with the data sources to retrieve data), while the latter two are service-specific for service discovery and service mediation (these queries contain information about agent services). More specifically, agents are seen as service providers, i.e. the Directory Agent provides semantic match-making services between queries and data resources, and the Resource Agent provides data-retrieval services etc. Task Agents and Resource Agents publish their agent descriptions with the metadata repository in a Directory Agent, using agent registration requests. Then, the other agents are able to locate the appropriate services within the system by querying the repository, via the service delegation query. Figure 13 shows how the message interfaces described above are used between UA, DA, RA, and TA.

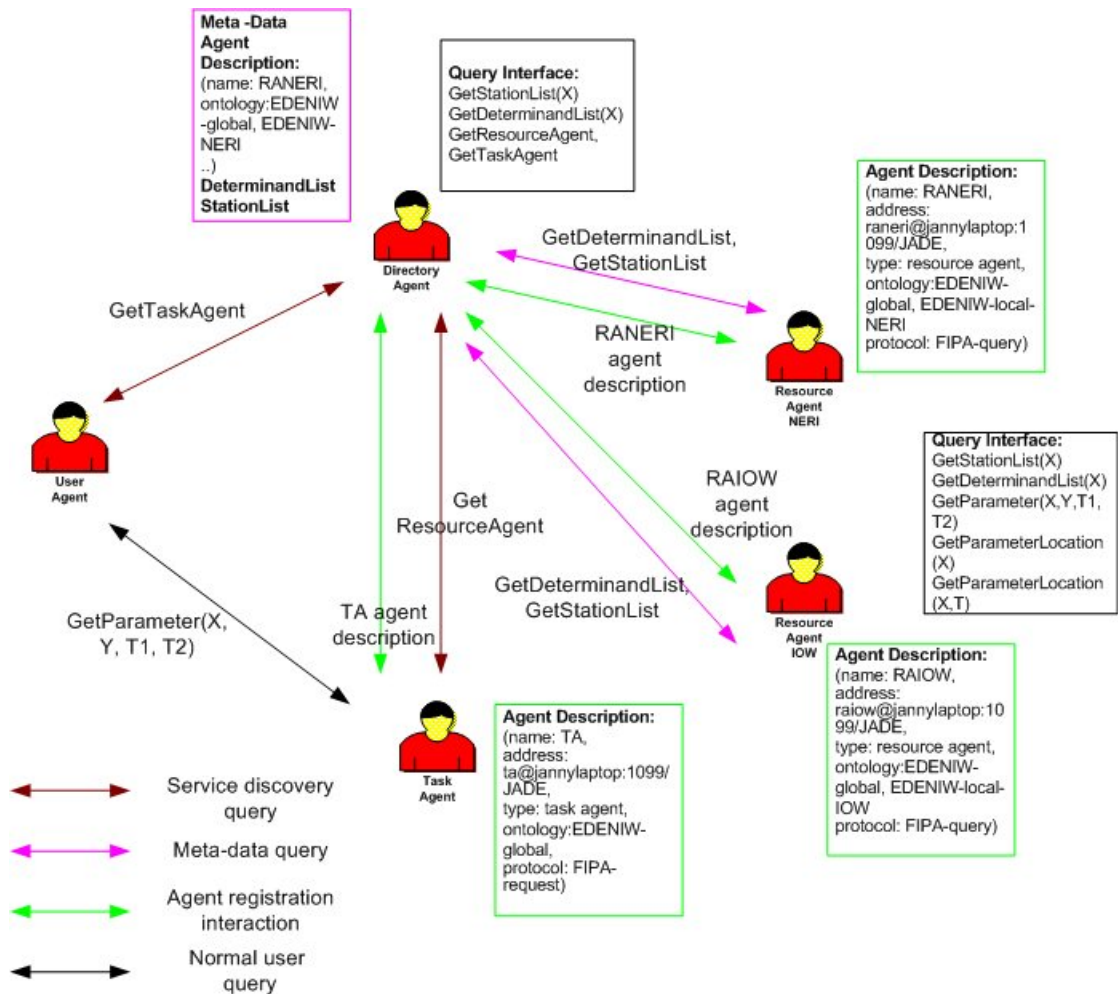


Figure 13 Agent interactions for metadata queries and agent registration and service discovery processes

An assumption is made that contact information in the Directory Agent is made public and can be retrieved from a configuration file. Task Agents and Resource Agents then register their agent description with the DA when they start up, to build up the metadata repository in DA. The data structure of an agent description is illustrated in Figure 11. When the DA receives a registration request from an agent, it updates its list of agents and an inform message is sent back to the requester indicating the success of the registration. Once a Resource Agent is registered, the DA pulls metadata from it periodically, i.e. sending out metadata queries, GetDeterminandList and GetStationList, and fills out the metadata repository according to the Resource Agent's reply. The DA has query interfaces of GetResourceAgent and GetTaskAgent for other agents to access the metadata.

4.4 Use Scenario

The agent interaction is depicted in Figure 14. Numbered arrows in solid lines reflect the interaction of processing a user query in general. The dotted lines refer to a metadata query.

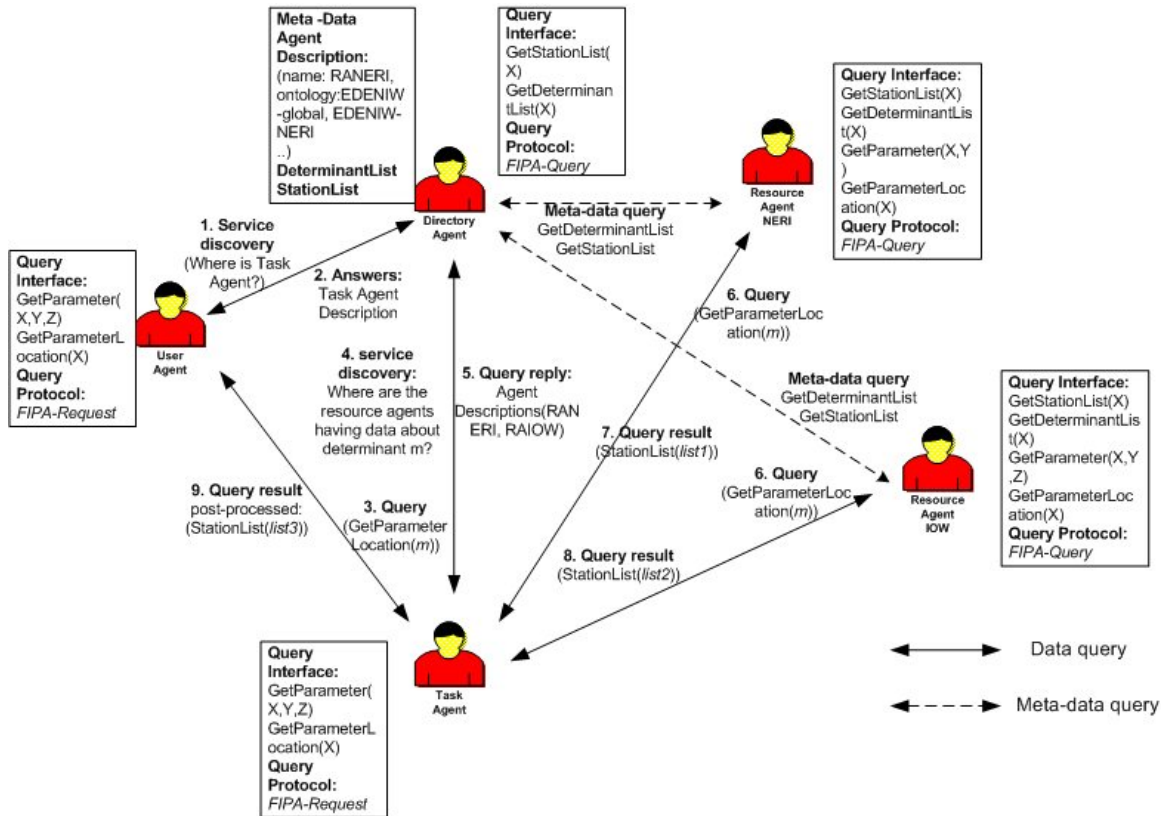


Figure 14 Agent Interaction for user query and metadata query

More specifically interactions in processing a user query in the EDEN-IW system consist of the following steps in normal situation, as opposed to the abnormal situation when errors occur:

1. UA (user agent) queries DA (directory agent) for contact information of TA (task agent);
2. DA replies with the agent description of any available TA in its directory;
3. UA contacts TA with user query, in this case GetParameterLocation(m). This means *give me all the stations that have information about parameter m*;
4. In receiving the query, TA analyses and extracts out key word parameter *m* and uses it to query DA to locate relevant data resources;

5. DA replies the metadata query with two agent descriptions, RANERI (Resource Agent NERI) and RAIOW (Resource Agent IOW) as both of them registered as having data about parameter m ;
6. TA sends the GetParameterLocation(m) to RANERI and RAIOW;
7. RANERI replies with StationList(list1);
8. RAIOW replies with StationList(list2);
9. TA combines the two results and replies to UA with a list of stations having data from two databases, i.e. StationList(list3).

4.5 System Performance Measurement

4.5.1 Descriptions of the Experiments

One of the main indicators of the efficiency of a query processing system is the performance, how quickly the results are reported to end users. As the key component of the query processing system, the Multi-Agent System's performance directly affects the overall performance. Some factors that also affect the performance are however beyond the control of the system, such as network delay and message latency due to the nature of the distributed environment. The processing time within agents is less dependent on the status of the network. The Task Agent is responsible for query distribution and result combination and thus acts as a key agent to measure performance. Therefore the query processing time at the Task Agent is selected to be measured. To test the inter-agent communication efficiency, the interaction between Task Agent and Directory Agent is selected, because the query processing within DA is independent of the network status.

The experiment's aim is to measure the query processing time at the TA and the message round trip time between TA and DA. The objective of this experiment is to collect performance data from the selected agents within the prototype system that will reflect the overall system performance. The statistics also indirectly reflects the efficiency of the query processing mechanism.

4.5.2 Measurement Results

In the experiment, the query processing times for DA and TA in handling the use case 1 core information queries are measured. The hardware configurations are a PC with an Intel x86 Family 6 CPU and 1MB RAM. As to software, the operating system is Windows 2000, and Agent platform used is JADE 2.6.

Table 4 Results of 5 times repeated UC1 query. Times in milliseconds.

Times	RDF Message Processing Time at TA
1	440
2	580
3	841
4	421
5	501

Table 5 Results of 5 times repeated UC9 query. Times in milliseconds.

Times	RDF Message Processing Time at TA	Round Trip from TA to DA, Transmission+Processing Time
1	390	421
2	451	971
3	331	440
4	361	1923
5	430	892

Statistics in Table 4 and Table 5 shows that the processing of RDF message can require a significant amount of time. Also this is a scalability concern when using RDF as a content language, because if the results size grows to tens of thousands, a RDF message might become too huge and costly to process efficiently. As well as the fact that using RDF as a content language gives benefits such as embedded metadata and eases semantic harmonisation of the results, the performance is a bottleneck. Researchers are working on optimisation algorithms to improve the processing on large RDF documents. Finding a balance between improving performance and imbedding huge amount of metadata is a key issue.

4.6 EDEN-IW Prototype Achievements

As part of the Queen Mary research contribution to the EDEN-IW project, the author made significant contribution towards the overall achievement on the project prototype. In particular, the author focused her contribution in the aspects of the design and implementation of a task / result sharing framework, the design and implementation of a RDF message interface that served as the main communication interface, and the design and development on the directory service. A summary on the advance information system is as follows.

1. An agent infrastructure is embedded within an open source agent framework based on FIPA standards. More specifically the agent platform used is JADE, whose design and implementation is FIPA compliant.
2. DAML+OIL was used to support seamless high level access to, and information exchange from, distributed data sources for four European water quality databases.
3. Design and implementation issues of using FIPA have been investigated, more specifically in the design of the directory agent and the extension of the RDF CL specification. The DA in the prototype EDEN-IW system expands the JADE DF functions, in that it records specific agent descriptions rather than simple name and address. In addition the DA pulls the RA status periodically to see if they are still alive. As to the application of the FIPA RDF CL specification, EDEN-IW has been a valuable experience in identifying the potential of RDF in capturing domain dependent action level logic (see section 4.2.5.2 for detail discussion).
4. The prototype demonstration was successful. In addition to the FIPA CL extension based on RDF0, the EDEN-IW also developed a service interface for agents based on RDF. This is mainly used for service discovery and matchmaking in the directory agent via the JADE DF facility.

4.6.1 Author's Contribution

The author's contributions in particular are the outlined as follows.

1. The design and implementation of the TA, which acts a broker between query issuers and data sources. The brokerage involves some meta-level agent interactions in order to know the status of an agent for example.
 - a. Query processing: decompose the query and forward sub-queries to relevant data sources, post-processing of the query results, combining results from different sources and simple operation on specific results such as the unit conversion.
 - b. Query management: keep track of each query session, report query processing status back to the end users, handle time-out and other exceptions during processing.
2. The directory services are wrapped in a directory agent, which utilises the JADE agent management facility to keep track of agents' online/offline status and to maintain a centralised metadata repository summarising data sources available in the system.
 - a. Agent status management: agents register their service descriptions with the directory agent when they first start off in the system. The Directory Agent periodically pulls the agents on its list to see if they are still alive.
 - b. Metadata management: data summaries are based on the key elements in the domain, determinand, station and time.
 - c. A public query interface on the metadata: both agent service descriptions and data source summaries can be queried. Only an exact match function is supported in the prototype, which means that a metadata query either replies with the exact information queried or nothing at all.
3. The RDF message interface basically specifies the formats for message exchange between agents, which includes metadata queries, data retrieval queries, agent registration messages and status report messages. The message interface closely relates how agents process the messages, therefore it links back to the internal behaviours of an agent to some extent. One of the main purposes of having a content language for FIPA is to separate domain specific knowledge processing with application specific ones. From this angle, it is particularly relevant that a good message interface helps the message processing.

4.7 Discussion

EDEN-IW project successfully achieved the core requirements, demonstrating a state-of-the-art information integration framework combining both agent technologies and Semantic Web, which is verified with the prototype system. The information querying and processing framework developed during the EDEN-IW project fulfils the query orchestration requirement, see section 2.1.4. However due to the pressure of producing a working prototype system within a fixed time-span and the recognised difficulties of distributed system development, the consortium had made some simplifying assumptions. In particular, Users are only allowed to ask certain types of queries. Other restrictions also exist, for example the agents are designed to follow a specific start-up sequence to ensure the directory agent is ready for provider agents to register their service descriptions at start-up. The system is therefore not very flexible. In addition, fault-tolerance was not made a requirement. It is assumed that system components are always available.

The search functions of the Directory Service are simple, and only exact match is supported. It is partly because the data summary about data source is one-dimensional name-value pair list containing all the determinand IDs and station names in a database. As the central metadata repository and metadata service provider, Directory Agent is likely to be a single point of failure or performance bottleneck. The MAS is thus fragile in the sense that the whole system is likely to halt if DA is not available. To have a more robust MAS infrastructure, it is necessary to have distributed local copies of partial metadata so that the failure in DA would not be disastrous.

Agents do not have memories of past events, so it's a stateless system. Agents do not make use of what happened previously to help adjusting their behaviours for the future. The system is closed because resources and agents are always assumed to be online, and the system lacks the flexibility to deal with addition and disappearance of agents dynamically. Stateless agent model might be sufficient in a closed world with specific set-up constraints. However it is far from enough for an open environment where components are more heterogeneous with more complex problem-solving features, and are free to join and leave the system.

Interactions in the prototype system focus on the query type only, i.e. most interaction uses the FIPA-Query protocol, which implies that the action is looking for an answer of a particular proposition, or in this case of information retrieval, it requires a set of the result as answer to the query. Interaction assumes collaboration, this cannot always be assumed in an open system. For more complicated scenarios such as handling an error in the system with cooperation from different agents need more dynamic and diverse interaction.

Another important issue for distributed information retrieval is data security. In some cases, the raw data is considered so sensitive that data owners may not want to share it with the public. Malicious queries need to be detected and filtered out before damaging the database. Those are the concerns from a data owner's point of view. There are security issues associated with information agents as well, for example an agent can be malicious and provide incorrect data deliberately to corrupt the system. In EDEN-IW system, protecting the raw data from malicious queries is given a higher priority than screening and isolating malicious agents. Agents in EDEN-IW are assumed to be benevolent, i.e. they will not provide incorrect replies deliberately.

To work around the security issue in EDEN-IW, resource agents are deployed locally at the data owners' site behind the firewall, and communicate with the rest of the agent system using HTTP. This model provides a certain level of security, because certain types of traffic are monitored and even blocked if necessary at the firewall. However it does not address the filtering of malicious queries, because this cannot be implemented at the firewall, but needs detailed analysis of the queries to detect any potential damage on the data set. This approach is explained in more detail in [49].

4.8 Summary

This chapter describes in detail the development of an information integration system in the inland water quality data domain, from analysing requirements, to design and implementation of the prototype system. One of the major limitations is the lack of support for an open service environment. Openness refers to the potential of interaction with unknown entities or services, as well as the non-deterministic service availabilities, i.e. services may come online and offline without a prior notice.

To be able to deal with open application environment, the core EDEN-IW prototype system needs to be extended to support flexibility and robustness. The flexibility concerns agent interaction patterns, service discovery schemes, and the use of semantic models to assist data analysis. Agent interaction flexibility refers to the ability to devise alternative responses to unexpected circumstances. A more advanced service discovery could enable service descriptions to be compared using multiple criteria, for example services can be searched by their capabilities, the quality of services provided and their availabilities etc.

Robustness is also a major concern to enable the system to continue to function with the presence of exceptions. Building fault-tolerance into the agent system is crucial especially when the environment is non-deterministic and prone to disruptions and exceptions.

Chapter five presents an enhanced MAS infrastructure for information retrieval that is more flexible, and the fault-tolerance of the framework is described and evaluated in chapter six.

5 EDEN-IW MAS Extension 1: Decentralised Directory and MDDV

5.1 Introduction

It has been proposed in chapter two that a hybrid MAS structure that encompasses web services and semantic web technology is the most suitable architectural choice to tackle the heterogeneous data and usage interoperability challenges in an open service environment. A hybrid MAS structure is able to provide a more flexible solution in terms of accommodating new applications and services whilst isolating the changes from impacting existing processes. Two main components have been added here to extend the framework of chapter four: a decentralised directory structure that also supports service and resource selection based upon quality of service and a semantic virtual data warehouse that supports multiple data views and derived data analysis.

It can be useful for agents to initiate interaction without mediators. This is particularly important in an open environment, because it is likely that agents may come online and go offline without prior notification to others. If mediators are always consulted for metadata queries about agents' contact details, query interfaces, and real-time status, it adds a significant communication overhead if managed purely by a central directory. It can represent a significant service access bottleneck and a single-point of failure for the system. In addition, user-mediator and user-provider interaction is often constrained to client-server, one-to-one, pull-type service interaction.

In typical directory service designs, the service capabilities are the only type of metadata available that users can use to select the services available. This has limitations. For instance, queries with strict time constraint require highly responsive service providers, which cannot be matched to any of the characteristics in the typical service capability metadata. Use of Quality of Service (QoS) data could enhance service discovery and selection on more complex user requirements. The QoS information needs to be obtained, published, queried, and managed. More

advanced service matching can then use not only services' published capabilities but also their QoS.

One final additional feature to increase MAS flexibility for use in open service environments concerns the ability of the metadata services support to virtualise data resources and its extension to support more flexible data analysis and to provide additional, high-level views of the data, e.g., to perform data warehouse Multi-Dimensional Data Views (MDDV). In this chapter, the use of a semantic metadata model extension is developed to support a virtual warehouse metaphor and this to promote more flexible, multiple, user views of the data.

The remainder of this chapter is organised as follows. Firstly, an application scenario is presented as a workflow showing how the main components collaborate with each other. Design issues are then discussed, focusing on a decentralised directory structure, service quality metadata to assist service discovery and selection, and analysis type metadata to assist high level data analysis. The MAS framework extensions are then evaluated.

5.1.1 Application Scenario

Consider the following scenario of how different services are typically orchestrated to answer a query. A user poses a data query “which river is the most polluted in year 2002 in all available countries” into the system, and initial processing suggests it is a high level Decision Support System (DSS) query, which requires analysis and post-processing along multiple dimensions including time and location. A search for the central directory is sent out, in order to find a service with multi-dimensional analysis expertise. The directory is not online, and the search fails. An alternative to use of the central directory for service discovery is to search any local metadata repository, were they available, to see which services have been used in the past for this kind of query, and acquire contact information from that. If this search again returns no result, then a message can be broadcast to other agents in the MAS to search for the right service. More specifically, a “Hello” broadcast message is sent to locate if a Multi-Dimensional Data View Service or MDDVS (“MDDVS request”) is available.

Consider the case where one MDDVS replies (“MDDVS1 capabilities available are ...”). Now that a service match has been detected, the original data query can be forwarded to MDDVS1. MDDVS1 decomposes the query into sub-queries, by consulting the domain ontology and multi-dimensional view ontology. It again finds the system directory unavailable for any metadata queries. It tests whether the last service it uses for task scheduling and allocation is alive with a “hello” request (“TS1 please reply if available.”). If TS1 is alive, a reply is sent. MDDVA then sends the sub-queries to TS1, who then takes care of the query distribution and result collection. The directory comes online, and TS1 queries the directory for metadata on which data sources to use. TS1 distributes the sub-queries in order, and sends the combined replies to MDDVS1 for any post-processing needed. During the whole process, a Monitor Unit keeps track of the agent conversations and analyses at a system level if any abnormal patterns occur. If there are any, an Exception Handling (EH) agent can be activated to deal with the exception. This is discussed in more detail in the next chapter, chapter six.

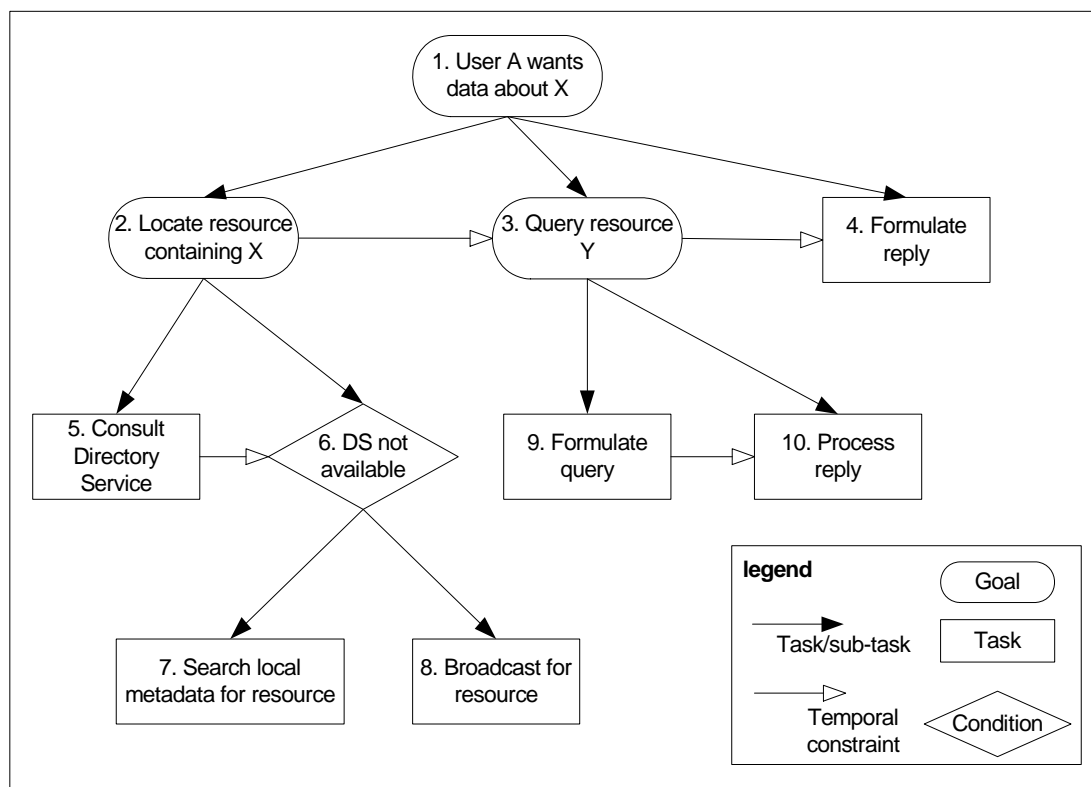


Figure 15 A plan for a user to retrieve a specific set of data

Figure 15 shows one plan instance for the goal “user A wants data about X”. There are three sub-tasks that consist of a plan to achieve this goal, namely locate resource containing X, query resource Y (that has data about X), and formulate reply. These sub-tasks are temporally constrained in the above order, because a query cannot be sent if the right resource is not located right at the start. In terms of service discovery, there are several options. There are at least two alternatives to use when a directory is not available: consult the local metadata repository, or broadcast to find a suitable service. Of course service discovery can vary in different situations. For example when a new user starts up, it may not have any entries for the local metadata, and it is more likely to prioritise using the centralised directory if available or to send a broadcast request to discover the services it needs.

Some data queries posed to the system contain specific constraints, such as reply within a certain time span. The efficiency of the query processing workflow depends on the latencies (round trip time for messages) of each service that compose the workflow. Generally speaking, network delay and message processing time are the two major factors for latency in distributed systems. If a query has a stringent time constraint, then query processing workflow should pick the services with least latencies.

5.1.2 Architectural Overview

In this chapter, the MAS framework of chapter four has been extended to support a decentralised directory service and an extended metadata services to support derived data user viewpoints. Web services specifications are used as an exchange data model to wrap up DBMS functions and to input data and user requests into the MAS. Semantic web languages are used to represent and model domain and application knowledge, and agents act as semantic mediators to interlink these components together forming a coherent framework that supports derived data analysis.

In terms of functionality, agents are equipped with a local metadata repository to enable alternative service discovery, and utility functions (defined in section 5.2.2.1) to determine the priorities of their actions. With respect to agent interaction,

broadcast within the agent platform is supported. A Hello Protocol has been introduced to facilitate initial agent interaction to obtain contact details from peers.

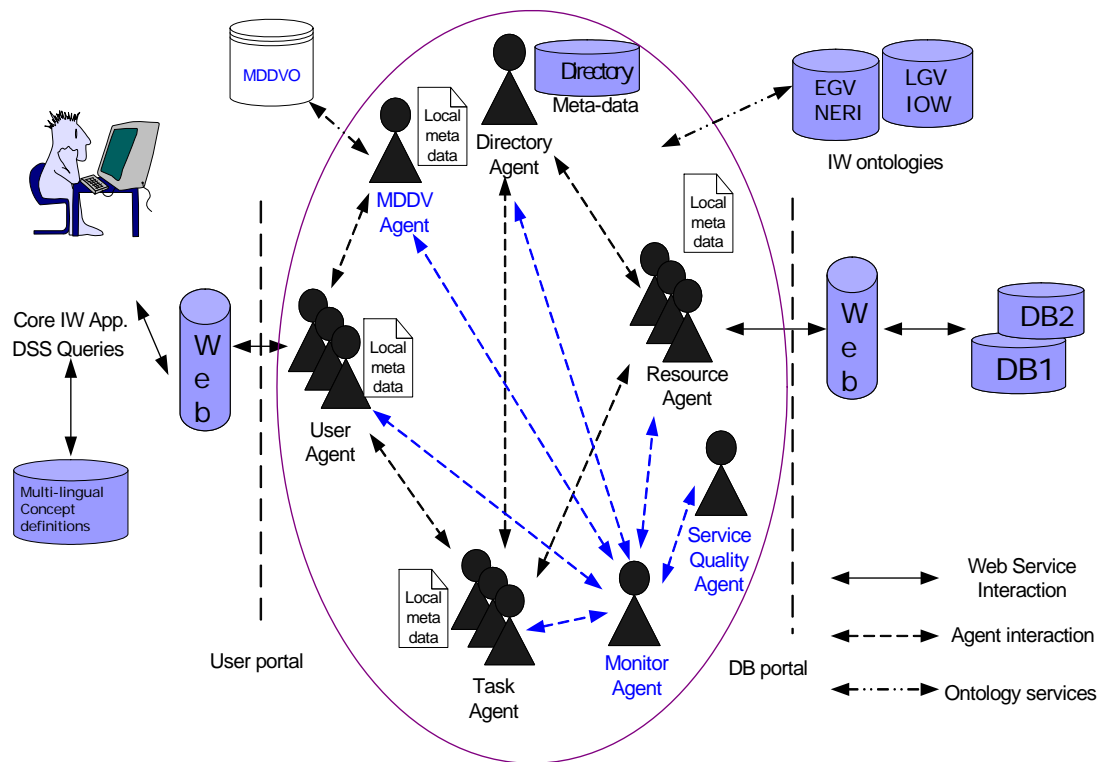


Figure 16 Architectural overview of an extended MAS to support an improved directory service and MDDV.

The main types of the agents and system components are depicted in Figure 16. Agents with names in blue are new, namely the Multi-Dimensional Data View Agent or MDDVA, the Service Quality Agent, and the Monitor Agent. Each agent also has its own local metadata service usage repository (some agents may not have the “local metadata” box next to them in Figure 16 for the succinctness). A Multi-Dimensional Data View Ontology (MDDVO) has also been added to assist the high level data analysis.

The key types of agent services defined for this extended system are:

1. User Agent (UA): represents the end user to formulate queries and display results in required formats. The UA has to select the right resources to send the query that best fits the query constraint.

2. Multi-Dimensional Data View Agent (MDDVA): decomposes complex data queries along multiple dimensions into sub-queries, and harmonises the post-processing data.
3. Directory Agent (DA): maintains and provides access to a central metadata repository of service capability descriptions and service status information.
4. Task Agent (TA): distributes queries into multiple data sources and manages multiple query sessions.
5. Resource Agent (RA): wraps a data source with agent features to provide uniform data access and more efficiently answer queries.
6. Service Quality Agent (SQA): maintains a repository of service quality data based on service invocations.
7. Monitor Agent (MA): keeps a record of agent interactions, to monitor the system operation and to detect anomalies if exist.
8. Exception Handling Agent (EHA): handles exceptions. Note this is described in detail in the next chapter, and therefore not shown in the architecture diagram in this chapter.

As before, see section 4.2.1 of the description of the architecture, all new agents communicate with other agents using ACL, and use RDF as the content language to represent service actions. The semantics for the MDDVA service terms are defined using OWL. Both the SQA and MA operate on operational metadata collected at run time.

5.1.3 Data Metadata and DSS Queries

The MAS system described in chapter five can handle three different types of queries: data queries, metadata queries and DSS queries. DSS queries here refer to those queries that cross multiple data sources. Metadata queries are used to enhance the smooth operation of the system with respect to changes and updates in data resource access, service discovery, and metadata management. Although handling data queries effectively and flexibly is the main focus of the framework, metadata queries are a vital enabler for these. The third type of query handling concerns DSS queries. Examples of each of these types of queries are as follows.

1. Low level data query (Use Case 1 query): what is the observation value for determinand X at station Y during time Z?

2. Metadata query (Use Case 9 query) and variation: which stations have measurements of determinand X? Which stations have measurements of determinand X over threshold Y?
3. DSS query: which river is the most polluted in year 2000? (See section 5.3.1 for more examples of DSS queries).

Metadata queries in turn can be further categorised into the following types:

1. Agent social query: Hello request, register service with directory.
2. Directory query: list all the stations available in a DB, list all the determinands in a DB, inform DB update.
3. Monitoring query: collect interaction metadata, collect service quality data.

5.1.4 Design Issues

To enhance the accuracy of service discovery and selection, service quality is introduced. There are several determinants for service quality, including performance, reliability and availability. The quality data is collected from a system-wide monitoring service that each agent has access to. More specifically, service performance includes latency and throughput. Latency has already been discussed. The throughput of a centralised service is also an important metric of service quality. The service quality metric builds up a useful profile of services, and affects the frequency of service invocation.

The data query proposed in the application scenario requires even more complex agent cooperation compared to the system given in chapter four when: searching for the right services, distributing tasks, collecting data and interacting. For example, a data query requires high-level analysis along several dimensions (temporal and geographical are two major ones) before results can be presented, e.g., to help policy maker users to assess the effectiveness of water quality management policies. These analysis functions are closely coupled with the IW domain knowledge, e.g., polluted as a water quality indicator may be defined in the water policy standards, and will need to be mapped into local database models because different data sources will most likely use different measuring units and represent different granularities of data. The generation of these analysis functions can be done on a per-query basis. Or the

generation process can be abstracted into templates a priori and instantiated for invocation whenever needed. The second approach is clearly more efficient, and it is the principle of the semantic view model proposed later in section 5.3 for flexible derived data analysis.

5.1.4.1 Using Gaia Agent Models

The original Gaia [125] modelling methodology focuses on the organisational view on the multi-agent system, and proposes the use of roles, resources and interaction to capture conceptual design of a multi-agent system. Roles are further defined by the responsibilities associated with them, functions, and permissions on certain resources. The MAS employed for an information retrieval application however, does not necessarily possess strong organisational features, and might be more sensibly centred on the notion of services. The Gaia model is adapted so service is the centre concept, instead of roles. There are two concepts associated with services: individual service actions or service (action) descriptions and service processes that define the patterns of service actions that are exchanged between a service client and provider. There are standardised service description languages (such as OWL-S), to define service interfaces in a reusable and interoperable manner. Quality of Service is defines how well and reliable the service performs. More specifically, the main properties of a QoS contain performance, availability, and reliability.

Roles are made to connect with agent interactions, in that every participant in an agent conversation assumes a certain role. The role in the conversation also helps identify the exact responsibility of the participant, therefore recognising the normative action associated with the role.

Agents are providers of one or more services, and the inter-agent interaction could be captured as service invocations. Services are the externally accessible functions of agents. The internal processing of an agent is represented as agent's goals and plans of tasks or actions to achieve these goals. In addition to the definitions of the key concepts, agent type, service, and interaction, some metadata models are included into the design model, such as service ontology, plan library, and interaction ontology. This can be seen as an extension of the original Gaia model, which does

not differentiate metadata layer concepts from data layer ones. Figure 17 shows these key concepts and metadata models. The service ontology defines the service description format, service quality properties, and their link to performance measurement. The plan library stores various plan templates for a number of situations, with the required parameters for instantiation. Agents choose one suitable template from the library, fill in with specific parameters, and a plan instance is generated for execution. As to the interaction ontology, it specifies what protocols are to be used in what situations, with termination conditions to ensure safe execution. The terminology used in the interactions is also defined in the ontology to avoid any ambiguity. More specifically, Agent Interaction Protocols, AIPs, are modelled as state transition tables for the different roles of the conversation, e.g. a request protocol is split into two transition tables, one for the request initiator and the other for the responder.

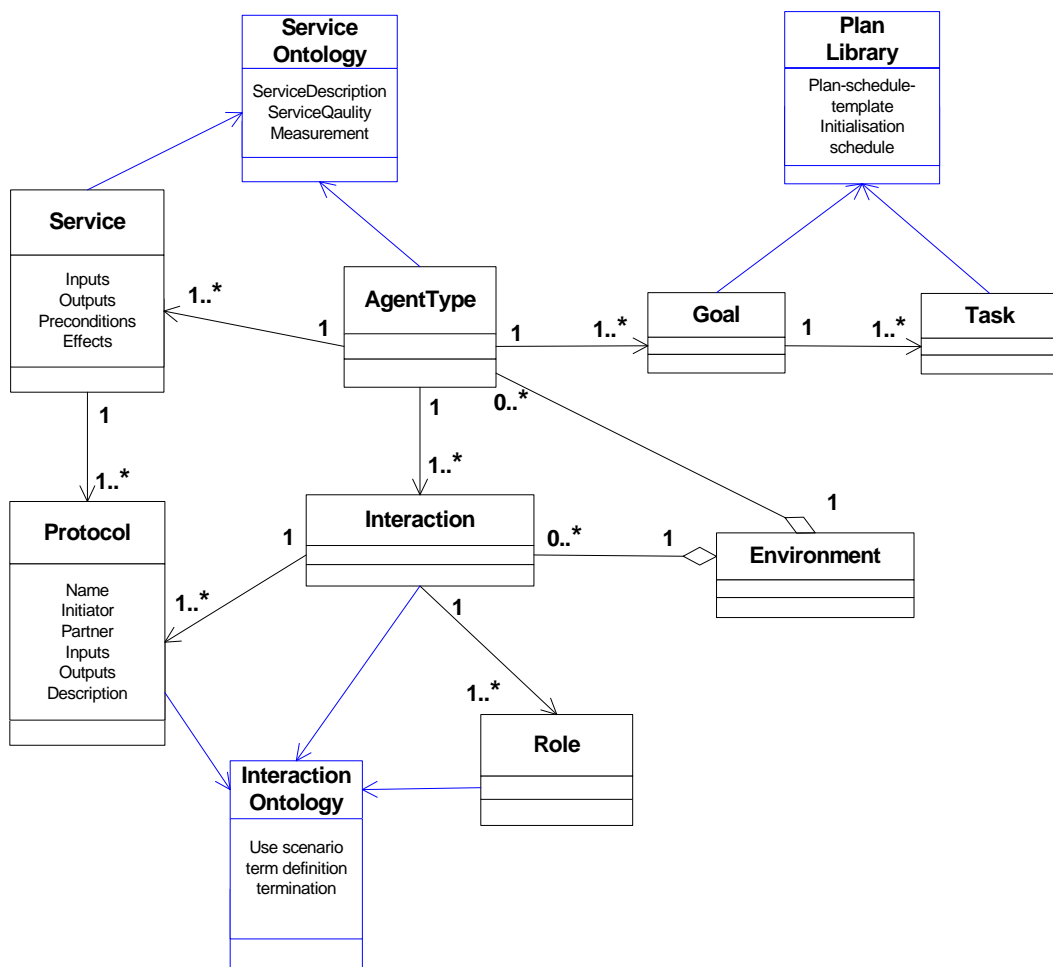


Figure 17 Key concepts in the conceptual layer of the adapted Gaia model

5.1.4.2 Modelling Agents versus Services

One of the main agent modelling choices is how to model services. Services could be re-designed and represented as agents or an already available service could be wrapped with an agent interface or it could be accessed using its (non-agent) service interface because the service interfaces and implementations are mature and are standard, e.g., the HTTP message transports service and RDBMS data management services. The added value of the service as an agent needs to also be balanced against non-functional factors, e.g., how more complex agent messages could degrade performance and affect security.

Services can be specified as followed.

1. Service category: classifies services of similar functions into types.
2. Service access: defines the level of access control a service implements, either public (accessible to all) or restricted (accessible to some). For restricted services, they might also include methods to acquire access.
3. Service description: defines the capabilities of a service with a profile model (as used in the OWL-S specification). The description also includes the interaction interfaces supported by the service.
4. Service quality: there are four properties of the quality concepts: availability, throughput, latency, and reliability. These properties can be translated into calculation of performance measurements, which are supplied by the system-wide monitoring service.

Service and Agent Interaction include inter-agent communication, service invocation, and interaction with system functions. Agent communication is inherently more complex than client-server service invocation, because it has to take into account the fact that agents are autonomous entities and have their own goals. The success of a service invocation simply depends on two factors, if the underlying communication link is working, and if the service itself is working. Agent interaction however depends on both of these two conditions as well as the agent's goals and plan execution to achieve them. For any interaction, there are participants, categorised by the roles they play, such as clients, servers, or mediators. The protocols these participants follow to orchestrate their behaviours and to achieve

common goals are another crucial part of the interaction model. Safe and sound protocols not only specify the expected outcomes of interactions under specific circumstances, they also indicate general procedures to follow in case of exceptions. For example, an Agent Interaction Protocol (AIP) for service invocation interaction should include specifications for exceptional conditions such as a user's needs to cancel a request underway, or for the provider to inform others that a failure has occurred, or when a third party monitoring service interrupts or suspends the interaction.

However when multiple agents with different roles and objectives interact and collaborate with each other, the message exchange becomes much more complex and requires system level coordination, it is more than simply putting together individual agents together.

5.2 Directory Service (Version2)

In the hybrid directory structure, the central directory service still exists to maintain all the published service capability descriptions from every registered agent in the system. However, the individual agents may also keep a local record of their interaction and processed messages. This then serves as their local metadata repository. There are overlaps between the metadata stored in the central directories and in these local repositories. For example the contact methods must be the same for a particular service no matter whether invoked using the local or centralised model service interfaces. However, the local model provides additional metadata not available in the central one, such as a specific user response time averaged across all user sessions or over a user's last and or current service invocation session.

In a sense, the central and local metadata repositories are complementary to each other. When sufficient, local metadata is used for service discovery, an external query to a central repository may be circumvented. However inconsistencies may emerge from these distributed repositories because each of them only has partial access to the overall situation and hence they may store partial knowledge that may not be coherent with the others. Mechanisms to check and maintain consistencies are designed and will be described in detail later in the chapter.

5.2.1 Ontology Model

The metadata in the system is categorised into three main types, service capability metadata, service quality metadata, and interaction metadata. The service capability information is supplied by service providers. The Directory Agent maintains a central repository of the service capability metadata, and individual agent each maintains the capability metadata of their contacts, i.e. peers they have previously interacted with. Inconsistencies may exist between the different copies of the metadata at the directory agent and individual agents, this issue is addressed in a later section. The Monitor Agent is mainly responsible for collecting the interaction metadata, and grouping the individual message exchange sessions that belong to the same conversation together. Further analysis can be performed on the interaction metadata, such as calculating the latency of the specific service session, and detecting anomaly in agent conversations. Service quality metadata is again obtained from further calculations on the interaction metadata, and maintained by the Service Quality Agent. For example service throughput statistics are obtained by counting the total number of processed service requests of a service provider during a defined time span.

5.2.1.1 Semantic Service Description

Services are described in terms of their capability and quality. The capability description takes on the OWL-S profile model, which in turn models services as processes. In addition to supplying services with their Input, Output, Precondition and Effects or IOPE, other information should be included in the description as well, such as service name, category, text description, and parameter. Also the accessibility makes up a key part of the service description, so that any party can check to see if it has access before invoking certain services.

Service quality is further defined with the following properties, availability, performance and reliability, which are clarified below.

- Availability: if the service is online and contactable.

- Performance: there are two aspects of performance: throughput refers to the number of service requests served during a given time period, and latency is the round-trip time between sending a request and receiving the response.
- Reliability: is specified as inversely proportional to a server's failure rate.

These attributes are calculated through performance measurements, i.e. a round trip time for a request, the failure rate for a service and the total number of requests being processed within a time period. These concepts are defined in the service ontology as well. The relations between the observation and quality attributes are modelled as constraints.

5.2.1.2 Service Quality and Performance Measurement Ontology

Figure 18 shows the ontological bindings of these service quality properties, and how their values can be calculated for different observation attributes. It is worth mentioning that some of the observations are instant values, while others are obtained from aggregation. For example, the request number refers to the total number of received or processed request within a time frame, thus it cannot be obtained at a single point in time. The inputs of latency calculations are however obtained instantly at the points where messages are sent and received respectively.

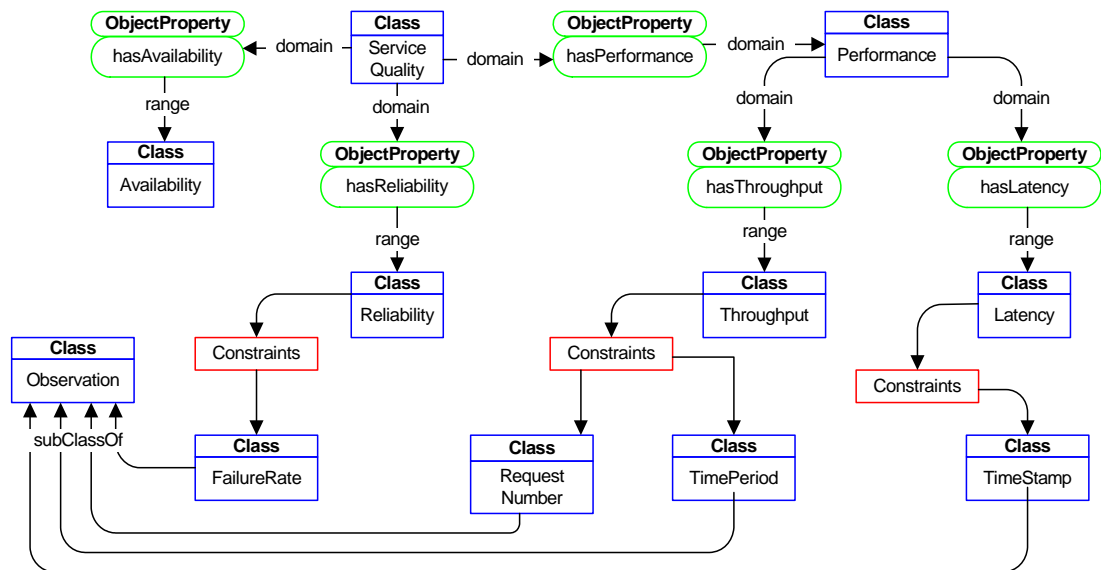


Figure 18 Ontology for service quality and performance measurement

5.2.1.3 Service Quality Metadata

Most of the performance measurements are provided by the Monitor Agent, for instance, the throughput and latency. Availability information is stored in the Directory Agent. Table 6 shows an example of the quality metadata on the service instance called RA1.

Table 6 An example of the service quality metadata

Service instance name	Message processing time (average)	Service up time	Last time of failure	Total failure count	Messages processed last minute
RA1	40 ms	50 min	NA	0	3

From the data on failure rate, the reliability can be calculated. Also throughput is defined as the total number of messages processed within a minute, so the message count is the throughput. The message processing time is calculated by dividing the total time used by all the messages one service processed within a period.

5.2.1.4 Service Description Based on OWL-S Service Profile Model

There are two types of service accessibility: public means every one can query, while restricted accessibility must also define possible methods to acquire accessibility. For example a service only accessible to group members, has an acquisition method to join the group.

<p>IWDirectoryAgentServiceProfile, subclassOf DirectoryAgentServiceProfile</p> <p>ServiceCategory=DirectoryService;</p> <p>hasProcess=AgentRegistryProcess</p> <p>hasInput=AgentDescriptionMsg;</p> <p>hasOutput=ConfirmRegistryMsg;</p> <p>hasPrecondition=none;</p> <p>hasResult=AgentRegistryEntryUpdated;</p>
--

Figure 19 A process model of the Directory Service profile

An example process modelled using IOPE is given in Figure 19 to describe the Directory Agent's process for registering agent services. As it can be seen from the example that the AgentRegistryProcess is NOT an atomic process, as the directory agent needs to check if the agent has been registered before, and if yes then DA retrieves the registered information for the entry and updates the record according to the new registration message, otherwise it adds a new entry into the registry for the agent. So it is a simple process, made up of two atomic processes, which are CheckIfEntryExistProcess and AddEntryProcess.

5.2.1.5 Query Representation

The query format here follows the basic structure of the one defined in Chapter four, i.e. a query specifies the query type, input values, expected output together with the processing status of the query. The query is also encoded in XML/RDF.

The EDEN-IW prototype system only supports two types of queries, metadata query and data retrieval query. Here more complex data queries such as queries that require data harmonisation and analysis are also supported. The RDF message interface in the prototype system needs to be extended to include rules in a message, to enable automation of query instance generation and increase the efficiency of message exchange.

5.2.1.6 Modelling Rules in RDF

Here is an example of the need for rule modelling in a query, e.g., processing query Q1 "which river is the most polluted for year 2000 in all available countries". The MDDVA decomposes it into two types of sub-queries: get names for all the rivers (SQ1), and get all data on stations where observation value of determinand X is over threshold Y for year 2000 in river Z (SQ2). The number of results of SQ1 obviously depends on specific data sources. However the number of SQ2 instances depends on both the results of the SQ1 and pollution definition. According to the Water Framework's specification (see EDEN-IW deliverable D18 [28]), several determinands' values are crucial in identifying how polluted the water is. The list of determinands with their thresholds is available in the MDDV Ontology. Consider a

case with four crucial determinands that define the pollution level, and 7 rivers in one DB, then there will be 28 (4*7) query instances for SQ2 for this DB. Instead of decomposing Q1 into 28 sub-queries for one DB, the MDDVA can include a rule for this sub-query generation together with the sub-query types and the list of determinands, thresholds, and rivers. The rule defines a repetition structure and the operation of replacing parameter values in SQ2 with each item in the determinand, threshold and river lists to generate new sub-queries. The rule is modelled in RDF is shown in Figure 20.

```

<rule>
  <repetition>
    <terminator ID=1>DeterminandList</terminator>
    <terminator ID=2>ThresholdList</terminator>
    <terminator ID=3>RiverList</terminator>
  </repetition>
  <operation>
    <operator>exchange</operator>
    <target ID=1>DeterminandID</target>
    <target ID=2>Threshold</target>
    <target ID=3>RiverName</target>
  </operation>
</rule>

```

Figure 20 An example of a rule to automate generation of queries

5.2.2 Internal Agent Design

Each agent has the following main internal components: message processing unit, task and planning unit, sub-task execution unit, and metadata management unit. Some of these units interact with corresponding metadata models defined in the conceptual design, for example message processing unit consults the communication ontology to decide what processing is needed. In addition, the execution unit interacts with the external monitoring function of the Monitor Agent that is responsible for overseeing the operation at the system level. Now that the Monitor Agent has the most up-to-date operational metadata reported from each agent, which forms the system operational metadata repository.

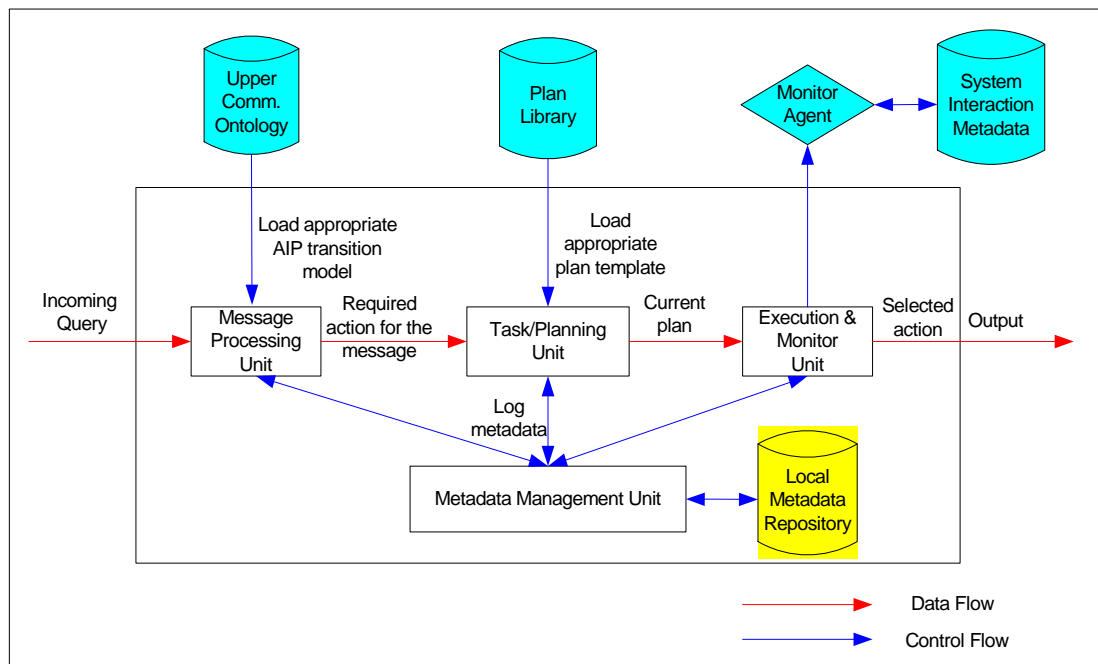


Figure 21 Data flows and control flow between modules of an agent

Figure 21 shows the data flow from coming into the agent’s message processing unit through to execution, and the control flow of the component interaction. After identifying the needed processing, current goal and tasks are evaluated to determine the priority for the pending actions. The task and planning unit then selects an appropriate template fit for the situation, fills it with the current parameters, and generates the plan. This plan is then fed to the execution unit for organised execution. As well as executing actions, the consequences and responses are monitored, so that discrepancies are detected in a timely fashion.

Meanwhile metadata of agent interaction and the message content are kept in the local metadata repository. The metadata management unit not only keeps a record of the metadata, but also performs active aggregation and analysis to make use of the metadata. For example, replies are linked to their requests so that latencies of the service invocations can be calculated. There is also an element of self-adjustment, in that whenever the monitor unit detects an exceptional circumstance it feeds back to the planning unit for modification of the current plan if necessary (which will be discussed in detail in the fault-tolerance model).

Table 7 and Table 8 show the examples of formats for agent interaction metadata and message content metadata respectively.

Table 7 An example of metadata information on agent interaction

CID	CA	Sender	Receiver	Time	ContentMetaID
001	Inform	RANERI	TA	2004112300103124	RANERI012U003

Table 8 An example of metadata on message content

ContentMetaID	Message Type	Search keyword	CID	Process Time	Result size
RANERI012U003	GetParameter	“POGN Y”	001	50ms	203 rows

With a local metadata repository, agents are enabled to make use of metadata of previous interactions to find services. There is also a message cache, caching all the messages in the currently active conversations for exception handling purpose (the detail usage is explained in Chapter six). When a conversation has terminated the messages are deleted so that the message cache does not grow into a unmanageable size.

5.2.2.1 Utility function

There are three main factors that affect an information retrieval agent’s utility: data completeness, process efficiency, and service robustness, in other words, an agent aims to provide its service in the most complete and efficient way as well as ensuring a robust performance. To maximise the utility, it would also choose services that are more reliable and efficient. So having service quality metadata increases the utility by choosing more efficient service providers, as opposed to selecting services only by their capabilities.

A utility function is devised to take into account the above three parameters. Completeness depends whether the service provider has the required set of capability, thus the value for completeness is determined by the capability match. Efficiency is reverse of the service latency, the longer the latency is the less efficient. Robustness is reverse of the failure rate. Suppose μ stands for utility, and c for

capability, l for latency, and r for robustness (reverse of failure rate), then the following formula denotes the utility function:

Equation 1: $\mu=r*(c+1/l)$.

5.2.3 External Agent Interaction

5.2.3.1 Interaction for Service Registration and Update

The Directory Agent maintains the registration information from agents and non-agent services. There are two ways the DA keeps the metadata repository up-to-date, by periodically pulling updates from registered entries, or by subscription to update notifications. Agents or services specify if they support update subscription in their service descriptions.

In case the DA is not available before any other agent starts up in the system, there is an alternative way to find and communicate with services, using the platform Directory Facilitator and to broadcast the request for a certain types of services. Agents establish ad-hoc connections initially, and the communication ultimately leads to the creation of individual agents' contact models.

5.2.3.2 Service Performance Monitoring

The SQA supports two main interfaces for other agents to provide measurement on service quality: register quality metadata for service instance, and update quality of service instance. It also supports searches based upon the metadata quality: get the quality for service X, and get all the service instances that fulfil quality requirement Y.

Performance measurements are normally supplied by the Monitor Agent. Upon receiving the individual quality metadata, SQA performs aggregation on the metadata. For example, average latency over a long period gives a more stable view on how a specific service provider actually performs.

5.2.3.3 Peer Interaction

The Hello Protocol is devised to identify if a party is interested in direct communication with the initiator. This specific communicative requirement is selected because it opens up the possibility to real peer to peer communications for agents. It means that agents no longer solely rely on directories, mediators, or brokers to find the agent with required expertise. The Hello Protocol is also more sophisticated than the Ping command commonly used within networking protocols. Ping is a very simple message to test if certain network connection exists, and it does not take into consideration any autonomy of the destination. Agents however, are known to be autonomous, which means they can refuse to reply. Hello Protocol makes the distinction between a refusal because the receiver is not interested and a failure because the message cannot be understood. It is important to distinguish these seemingly small differences, as they are crucial to model an agent's autonomous characteristics.

The scope for this general purpose introduction protocol as its name suggests, is to facilitate new entries to the system to initiate communications with other system components. There are two scenarios where the Hello requests are used:

1. It is used as a broadcast Hello request to search for a particular type of agent, i.e. the directory agent.
 - a. New entry agent X wants to contact DA, and broadcasts a Hello-Msg(type=1, content="Hello, DA please reply.", CA=request, time-out=30sec).
 - b. Only DA replies, while others ignore this.
 - a. When no replies arrive before time-out threshold, X assumes that DA is not available from the broadcast list.
2. It is a Hello request aiming for specific recipients, to exchange service descriptions.
 - a. X sends Hello-Msg(type=2, content="Hello, I'm X (x@apc)", CA=request, time-out=25sec, onto=sd-onto).
 - b. Now receivers decide if they want to reply according to individual request.

- c. Expected replies include the name and address of the replier, and an access method to use sd-onto.

A formal description of the Hello Protocol in AUML is given in Figure 22, which shows the flow of message exchange. The participant can reply to a Hello query with refuse, failure, not_understood coupled with a request (for a reformed query), and inform. All of these are FIPA standard Communicative Act or CA. The synthesised CA with not_understood and request represents the case where a clarification on the Hello query is needed. When this specific sequence of CAs is received by the initiator, it realises it is a request for clarification. The initiator sends an inform message first to confirm that the request has been accepted. It then sends the reformed query. The protocol terminates at the following three conditions: the participant refuses to reply the Hello query, he participant fails to reply, and the participant replies with an inform to the query.

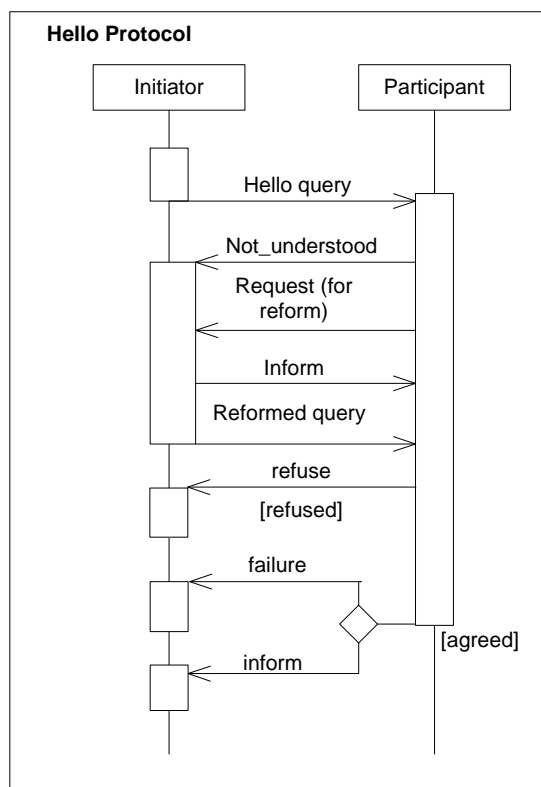


Figure 22 A Hello Protocol in AUML [2]

Some assumptions are made before the Hello Protocol can function in the system. These are outlined below:

1. There is no central point of connection, but directory agent is a logical choice as it maintains a centralised repository of agent descriptions.
2. A Service Description ontology (called sd-onto) is shared, which contains the information necessary for any communication involving service descriptions.

5.2.3.4 Monitor Interaction at the System Level

Individual agents only keep tracks of the conversation they are involved in, therefore they lack the ability to oversee the conversation's integrity at the system level. The Monitor Agent (MA) is introduced for the purpose of watching agent interactions at a conversation level and to detect any possible anomalies. Table 9 shows the metadata entries for individual interaction being collected by the monitor agent into a common format for further analysis.

Table 9 Aggregated Interaction Metadata

CID	CA	Sender	Receiver	Time	ContentMetaID
001	Inform	RANERI	TA3	2004112300103124	RANERI012U003
003	Query	DA	RANERI	2004112300104020	DA001U004
001	Inform	RAIOW	TA3	2004112300105002	RAIOW002U005

MA then applies analysis rules on the aggregated metadata, to single out specific conversation and to see if any exception occurs during the process. An example of all the metadata relevant to a specific conversation is shown in Table 10. The data shown in Table 10 suggests a possible anomaly in conversation 001, because there is a lack of an inform used by TA to UA1, the initiator of the conversation. This detection however does not always lead to error handling, because the lack of a response is not sufficient to say that something is wrong in the process. It might be that the reporting message from TA to MA is lost due to some transient messaging fault, which does not mean the actual reply from TA did not occur. The MA can choose to wait for a certain time to see if any confirmation of the reply message

arrives within the period, or it can actively ask UA1 if it had received the reply from TA3 on conversation 001.

Table 10 Metadata about interactions for conversation 001

CID	CA	Sender	Receiver	Time	ContentMetaID
001	Query	UA1	TA3	2004112300095859	UA001U021
001	Query	TA3	RANERI&RAIOW	2004112300103020	TA003U001
001	Inform	RANERI	TA3	2004112300103124	RANERI012U003
001	Inform	RAIOW	TA3	2004112300105002	RAIOW002U005

5.2.4 Matching Services based upon both Capability and QoS

The capability property describes what the service does, and the quality shows how well the service performs. These descriptions are defined in section 5.2.1.3. The directory service then searches service based not only the published capabilities but also on QoS. The search of services according to capability is performed by the directory agent, because it has a centralised repository of all the registered services' profiles. A capability match determines if the required functionality can be matched by the IOPEs of a particular service or a combination of services. Search by capability can only return a list of service instances of the same type that is capable of providing the service required.

The quality of service is however provided by the Service Quality Agent, where performance measurements are supplied by the Monitor Agent, and availability information pulled from Directory Agent. The Service Quality Agent returns the service instance that fulfils quality requirements, which narrows down the results from the capability matching.

Users decide if querying service quality metadata is necessary for service selection, some users might just want to pick the first service instance that is capable of the required service.

5.2.5 Service Discovery

Services need to publish their descriptions so that they can be searched. The service discovery process includes a publishing stage and the actual searching stage. At the publishing stage when an agent first starts up, it registers its name and address with the platform Directory Facilitator; the new agent then searches the DF for directory agent. If the directory agent is not available from this search, then the new agent can opt to send out the message periodically, or it assumes that when a directory agent comes online, it will send out an announcement message. Either way, when the agent gets into contact with the directory, it registers its service description.

At the searching stage, an agent can choose between three options in service discovery, query the Directory Agent and Service Quality Agent, query local metadata repository, or broadcast for the service.

5.2.6 Maintenance of the Local Metadata Repositories

Local metadata repository includes the service providers' contact details and message-related or interaction metadata. These local repositories are regularly modified to comply with the metadata from the directory agent. In terms of the maintenance of the contact metadata, the capability information of a service provider that can be modified according to later replies to the metadata query from directory, assuming the directory holds the most up-to-date information on service capability descriptions. Performance information is either modified from the agent's local measurement or informed from the MA.

When taking into account potential exceptions, it is more complex to synchronise service quality metadata. Because agents only have a limited perspective of the system and sometimes cannot see the overall situation, knowledge may become inconsistent. For example if a message is lost in the communication link, then the sender would have the message down as sent while the receiver would not have any record of receiving such a message. Conflicts like these can only be detected at the system level, i.e. Monitor Agent detects anomaly at the conversation level when

metadata are aggregated and analysed, see section 5.2.3.4. Then MA sends out an inform message to the participants of the conversation about the possible communication link error. Agents upon receiving the inform message can change their local repositories to reflect the actual situation. Quality measurements of the relevant services have to be modified accordingly. For example, a missing reply to a sent request could be interpreted as an unreliable receiver. However the actual cause of the missing reply is a communication fault. Then the rating of the receiver's reliability has to discount this failed request.

5.3 Semantic Multiple Dimensional Data View Design

The term derived refers to data that is aggregated or calculated from the raw set of data. One of the main application requirements comes from answering Decision Support System (DSS) type queries. OLAP on data warehouse is among the most commonly used techniques for high level data analysis, see chapter 2. OLAP is typically used with Multi-Dimensional DB or MDDB, in which data are represented by a multidimensional data structures. Here the derived data mostly revolves around finding a flexible design for MDDB that can assist DSS query processing. The design of a multi-dimensional data model requires a different methodology from the relational model. The method proposed by Thomsen [121] is as follows.

1. The starting point is to identify what must be tracked (for water quality data measurement depend on determinands). A collection of tracking variables is called a variable dimension.
2. The next step is to consider what types of analyses will be performed on the variables dimension. In a water quality monitoring system, water quality may be correlated over a period of time, e.g. this year from last year, or over one or more regions. These types of analyses cannot be conducted unless the instances of each variable have an identifying tag. That is, in this case, each measurement must be tagged with time and location. Each set of identifying factors is an identifier dimension.
3. Variables and identifiers are the key concepts of multi-dimensional database design. Some identifiers may follow a regular pattern, variables usually do not. For example the temporal identifier follows a sequential pattern from

days, months, quarters to years. The measurement variable may not follow a pattern.

4. The next step is to consider the form of the dimension. There are three types of variables, nominal, ordinal, and continuous. A nominal variable is an unordered category (e.g. region), an ordinal variable is an ordered category (e.g. time), and a continuous variable has a numeric value (e.g. measurement value). According to the types of variables and dimensions defined, different analyses can be applied, see [121] for more details. For instance, if the identifier dimension is either nominal or ordinal and the variable dimension is continuous, then analysis of variance is applied.

Following this method, several dimensions for the water quality data are identified and specified and the multi-dimensional database is subsequently designed. Another consideration is to incorporate semantic models into the process of data analysis, and to exploit where the semantic model can be used to enhance the flexibility.

Here the proposed virtual data warehouse is realised using data views. A view is a virtual or logical table in a database that is the result of a query. Because a view is not part of the physical schema, it is dynamic. A virtual table is computed or collated from data in the database. Views are much more flexible than tables, because they can subset data, join tables, aggregate tables with aggregated data, and hide complexity. Furthermore views do not require extra storage.

Sets of views can be derived from the core set of data queries in the problem domain, and templates can be generalised. Because the aim of the view approach is to hide integration complexity from the end users, so that view templates have to use terminology in the global ontology. The templates have to be translated from queries with global terms to SQL with local terms specific to each data source. The translation involves term substitution, table location, and constraint mapping. Some of the translation can be automated because there is one-to-one mapping between concepts in the global and local ontologies. For others the translation relies on semantic relations between terms and the rules that govern mappings.

This virtual data integration into a common repository approach is beneficial in that the integration process has been pre-processed before the data queries actually arrive, thus improving performance over the approach where query translation and mapping are done on-the-fly.

A multi-dimensional data view (MDDV) model has been applied to the IW domain data. Ontologies are used to model the semantics between these dimensions and hierarchies, and interconnection to the IW domain knowledge. The MDDV model design starts by identifying the set of data queries and the associated multi-dimensional analysis of interest. The semantic model for multi-dimensional data analysis (Multi Dimensional Data View Ontology or MDDVO) is then built and concept mappings between MDDVO and IW Global and Local ontologies are constructed using the already existing IW global to local mapping and associated constraints. Then two view templates are generated to capture the most commonly used queries, and the workflow showing template instantiation is given. A query is selected as an example to illustrate how the semantic multi-dimensional view approach works, going through the query decomposition process, and the generation of the view SQL query by filling in parameters for the selected view template.

5.3.1 DSS Data Queries

A set of data analysis queries are identified as useful for end users to have a high level understanding on the water quality (known as List 5.3.1 hereafter):

1. Which river is the most polluted for year 2000 in all available countries?
2. Which region in France is the cleanest for year 2002?
3. What is the mean value for determinand BOD5 at river RHONE for the first quarter in 1999?
4. What is the annual mean concentration for Nitrate at river SEINE for 1998?
5. Which region has the most stations with observation value of determinand “total phosphorous” larger than the threshold 500 µg P/l in the past 2 years?

5.3.2 Multi-Dimensional Database Structure

Queries use at least three dimensions for analysis, the time, location and the type of water quality indicator. The decisive variable for water quality is the determinand, defined as a physical, chemical or biological parameter or substance. So a third dimension in analysis water quality data is water quality itself, with varying levels of granularities. At the lowest level, this deals with single determinands, and the highest level consists of concepts such as clean and polluted. According to the design methodology for multi-dimensional databases described earlier in section 5.3, the measurement is the variable dimension, and time, location, and water quality are identifier dimensions.

From the analysis and processing needed in the domain (see List 5.3.1), each of the identifier dimensions is further divided into levels, as shown in Table 11. The decomposition of the time and location dimensions is relatively straight-forward, for example a year is made up of quarters and a quarter is made up of months. Decomposition of the water quality is more complex, because an observation is not only related to the determinand being measured, but also closely affected by other factors such as the medium the measurement is taken, the analytical fraction associated with it, and the unit used to represent the value. These factors collectively provide the semantics for observation values, and decide how to interpret them. For instance, the same observation value of pH in organic and inorganic analytical fractions would mean differently, maybe an organic analytical fraction has a maximum pH value less than 14. The example values for these sub-levels in identifier dimensions show that they are all either nominal or ordinal types. The variable dimension (measurement value) is numerical. Therefore the process applied to this multi-dimensional database is analysis of variance, e.g. comparing measurements by region, or by year etc.

Table 11 Dimensions with levels and instance examples

Dimension	Level	Instance
Time	Year	1999, 2000
	Quarters	first, second, third and fourth
	Month	January, February
Location	Country	France, Denmark
	Region	North-East, North-West, South-East, South-West
	River	“SEINE”
Water quality	DSS ready	Clean, polluted
	Harmonised	Units changed
	Analysed method	pH value of organic (analytical fraction) sediment (medium)
	Individual determinand measurement	Observation value of Nitrate

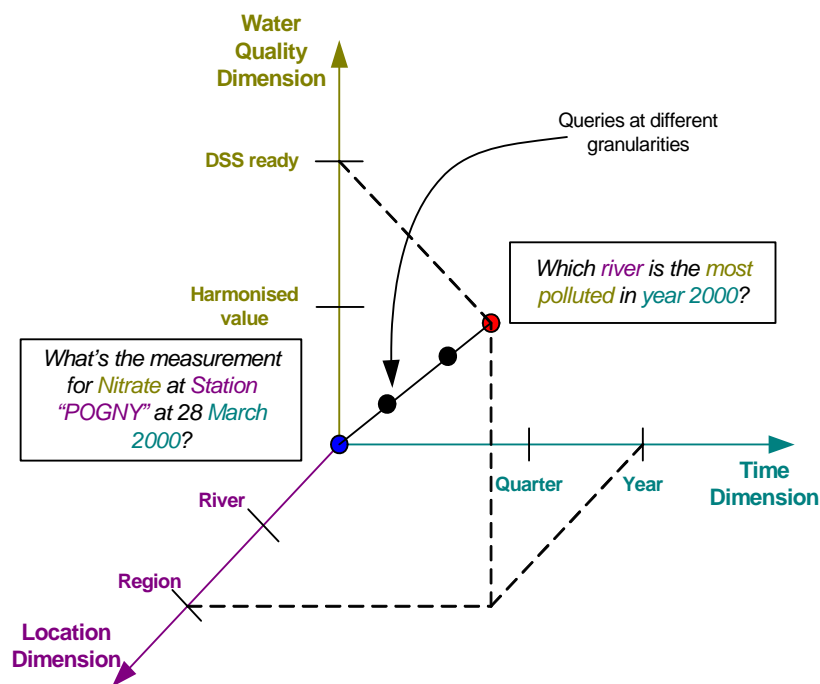


Figure 23 Dimension relations and query examples

Figure 23 shows how the identifier dimensions inter-relate and indicate the different query granularities using the various sub-level values. Each query in List 5.3.1 can

be decomposed along the three dimensions defined. For example the query “what is the annual mean concentration for Nitrate at river SEINE for 1998?” can be decomposed along the location dimension, to extract all the stations of river “SEINE”, and then low level data retrieval queries are posed upon these stations. In this case, the order of the sub-queries is crucial, because the results of the first sub-query are one of the inputs for the second sub-query.

5.3.3 Query Decomposition and Post-processing

5.3.3.1 Decomposition According to Dimensions

Consider the first query “*which river is the most polluted for year 2000 in all available countries*” for example, the analysis can be done respectively on the above defined dimensions. On the time dimension, *year 2000* is the constraint, which means that observation values throughout the year would be used to calculate a mean annual value. On the location dimension, *river* is the one to be assessed. So stations are grouped into the rivers that they belong to and upon which relevant post-processing is carried out. On the water quality dimension, *polluted* is the assessment criteria. The water quality *polluted* can be found in specific Water Framework Directive (WFD) defined in terms of observation values of particular determinands with thresholds (see EDEN-IW project deliverable D16 [28]). An example of the classification is shown in Table 12.

Table 12 River data quality classification

Determinand	High	Good	Moderate	Poor	Bad
BOD (mgO ₂ /l)	<1	1-2	2 - 3,5	3,5-5	> 5
COD	< 10	10-20	20 - 35		
Total Phosphorous (µg P/l)	<10	10-25	25-125	125	> 500
Orthophosphate					
Nitrate	< 0,1	0,1-0,3	0,3 – 1 (2,5)	1 (2,5)–7,5	> 7,5

According to Table 12, the main determinands that define water quality are BOD, total phosphorous, and Nitrate, each with several threshold values for different levels of quality definition. Observation values over the thresholds on each of these determinands need to be retrieved for individual stations. The post-processing then has to group data according to the river that station belongs to, and decisions are made based upon the comparisons between rivers. Generally the time dimension input is used as specific SQL constraints to specify the measurement time. The location dimension input can be translated into certain type of aggregation on individual measuring stations. The water quality dimension requirement however, depends more on predefined processes and help from the WFD to decompose a complex concept such as “polluted” into sub-queries concerning a set of determinand measurements whose values exceed defined thresholds. In this sense, the water quality dimension mainly deals with aggregating determinands and data harmonisation.

The query is then decomposed into the following sub-queries:

1. Get the names for all the rivers.
2. For each of the determinand (BOD, total phosphorous, and Nitrate) and for each river, get every station with observation value over certain threshold along with other information (such as determinand ID, unit ID etc.) within year 2000.

5.3.3.2 Post-processing of Results

Post-processing generally consists of collecting results from distributed data sources, data harmonisation, and transforming data into different presentation formats. Data harmonisation in itself is a complex process, which may involve several kinds of operations, convert data in one unit to another, multi-lingual translation, temporal projection, and interpretation of missing data. Also post-processing sometimes is associated with how the query is decomposed, and the order of the sub-queries. For example for the query decomposition in the previous section, the results of the second sub-queries need to be aggregated according to the results of the first sub-query, i.e. observation values need to be aggregated according to the river the stations belong to.

5.3.4 Semantic Model for Data Analysis Application

Two ontologies had been developed within EDEN-IW prototype for the IW domain knowledge, a global view ontology containing global concepts (i.e. concepts applicable to each data source connected to the system), and a local view ontology containing database specific concepts [129]. The multidimensional analysis uses new concepts such as “river” and “clean” and the application requirements pose constraints on these concepts, which form an application ontology on top of the EDEN-IW global ontology. Figure 24 shows the overlaps between MDDV Ontology (MDDVO), IW Global Ontology (IWGO), and IW Local Ontology (IWLO).

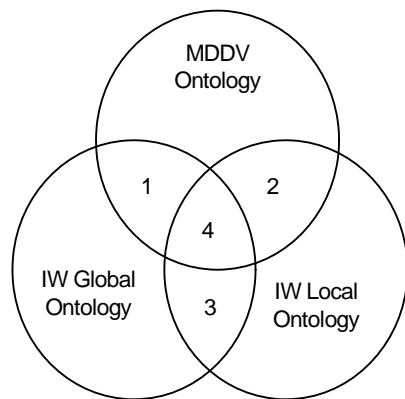


Figure 24 Overlaps of ontologies

There are four types of overlaps, see areas numbered 1, 2, 3, and 4 in the figure.

1. $MDDVO \cap IWGO \cap (\neg IWLO)$: Concepts exist in both MDDVO and IWGO but not in IWLO.
2. $MDDVO \cap IWLO \cap (\neg IWGO)$: Concepts are common to both MDDVO and IWLO but not IWGO. An example of such a concept is the catchment area.
3. $IWGO \cap IWLO \cap (\neg MDDVO)$: Only exist in the domain ontology, but is not needed in MDDVO. Community is an example of such a concept, because end users (except water domain experts) are not very likely to be interested in this level of detail.
4. $MDDVO \cap IWGO \cap IWLO$: defines common concepts used in all three of the ontologies. Station is an example that is defined in all three ontologies.

These overlaps to some extent determine which the direct mappings between concepts in MDDVO and IWLO are possible. For example concepts in overlap area 1 basically do not have any equivalents in the local ontology, which mean that either the concept is not captured in the DB schema, or the concept is represented with combinations of concepts with specific constraints and these constraints cannot be directly translated into combinations of concepts in the global ontology. In this case, the mapping can only be done with the help of a domain expert to generate a direct or indirect translation between concepts. That is also to say, a query containing such a concept is not answerable unless the mapping exists. As to concepts in area 3, they are less important, simply because a query with these concepts would not be asked by the users. Now for concepts in overlap area 4, there are equivalents in both IWGO and IWLO, and direct concept/term mapping can be readily drawn from the existing IWGO and IWLO. For queries using concepts from area 2, they are answerable because links between MDDVO and IWLO can be established even without the participation of IWGO. Here the mapping is represented as constraints, where the definition links concepts in the MDDVO, and instantiation consists of terms in the IWLO. It is worth mentioning that these constraints are inevitably DB specific.

Figure 25 gives an overview of a three layer semantic model that is designed to ease maintenance. It means that it is not necessary to have a global concept in between the concepts in MDDVO and IWLO. The overlaps numbered 1 and 3 in Figure 24 are not included in the semantic modelling, either because it needs expert input (for type 1), or it is not interesting to end users (type 3). It can be seen that “river” is modelled in the MDDVO, because the concept river is not explicitly modelled in the Danish DB (NERI) even though it may well be one of the most common concepts logically. Because it is not applicable to every single data sources linked by the system, the concept river is not a term in IWGO. In NERI, queries about river information are unanswerable without major remodelling of the semantic model. However it is explicitly modelled in the French DB (IOW), which enables the direct mapping from river in MDDVO to the IOW data schema.

There are two types of mappings from the MDDVO to IWLO, as mentioned previously. One is to reuse the already existing links between IWGO and IWLO. For example decomposing “river” at the location dimension gives a set of “stations”,

which exist in both IWGO and IWLO. However “river” is only explicitly modelled in the IOW DB, and not in NERI DB. Therefore “river” in the MDDVO is only applicable in IWLO-IOW. Likewise the concept of “catchment” is explicitly modelled in NERI but not in IOW.

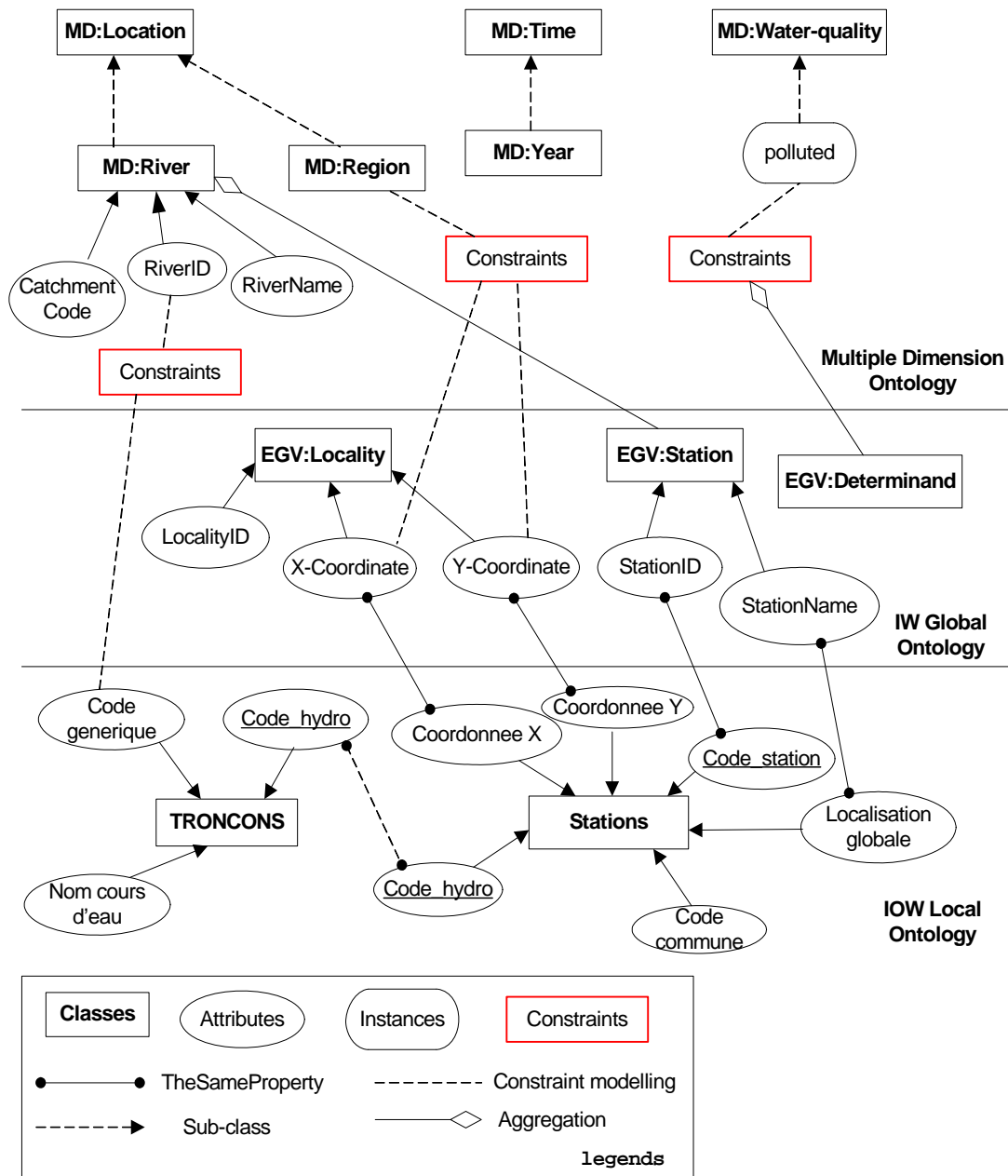


Figure 25 Three layer semantic model: multi-dimensional view ontology, IW global ontology and IW local ontology of IOW

For those concepts that have no direct equivalents at the global ontology, concepts are modelled in MDDVO and mapped to IWLO directly. The concept “river” is a

good example, and many DSS queries require data about rivers, however it is explicitly modelled in IOW DB but not in NERI. Therefore it is not modelled in the IWGO, and the mapping from MDDVO to IWLO-IOW needs to be modelled as a constraint. However queries about river data on the NERI DB is not directly applicable, and catchment is used as a similar concept and a substitute query is suggested to the users.

The following query decomposition analysis is based on IOW DB. Concepts such as station, determinand and locality are common in both global and local ontologies, and Table 13 shows some direct concept mapping from the IW Global Ontology to tables in the NERI and IOW DB schema.

Table 13 Table mapping between global, NERI and IOW

Global	NERI	IOW
Station	STBE_STATION	STATIONS
Determinand	KO_PARAM	Par metre
Locality	LOBE_LOKALITET	Troncons

River-related attributes are modelled in the MDDVO, and Table 14 shows the direct field mappings from MDDVO to IOW DB.

Table 14 Direct mapping of fields related to river from MDDVO to IOW DB

MDDVO	IOW
StationID	STATIONS.Code_station
StationName	STATIONS.localisation_globale
RiverSegmentCode	STATIONS.Code_hydro
RiverID	TRONCONS.code_generique
RiverName	TRONCONS.nom_cours_eau

5.3.5 View Templates

The essence of a virtual table is to wrap any data queries that can be formulated into an SQL select sentence into a new table, and the new table acts as a fact table upon which queries can be drawn. An example SQL sentence for creating views is *CREATE VIEW view-name (column-name-list) AS SELECT field-name-list FROM table-name-list WHERE condition.*

Analysis of the queries given in section 5.3.1 provides an insight to specify some commonly used view templates, such as grouping stations according to the rivers. In addition, it is worth wrapping the lowest level of data query into a view by itself, because many more complex queries are decomposed into this low level data query.

Two templates for virtual tables are designed:

1. A View on individual station's observation values: `CREATE VIEW view_observation (ObValue, DeterminandID, DeterminandName, StationID, StationName, UnitID, AnalyticalFractionID, MeasureTime) AS SELECT local_ObValue, local_DeterminandID, local_DeterminandName, local_StationID, local_StationName, local_UnitID, local_AnalyticalFractionID, local_MeasureTime FROM (joined tables that contain data about measurement, determinand and stations) WHERE (tables are linked by foreign keys);`
2. A View on observations at a particular river: `CREATE VIEW view_river_observation (ObValue, StationID, RiverName, DeterminandID, MeasureTime) AS SELECT local_ObValue, local_StationID, local_RiverName, local_DeterminandID, local_MeasureTime FROM (joined tables that contain data about measurement, determinand, stations and rivers) WHERE (tables are linked by foreign keys).`

Data queries can be answered using these two views with minor variations on the selection constraints. To instantiate these views, information on local column names and which tables to join are needed. These can be extracted from the specific database schema that the instantiated views are to be executed on. With the semantic models, the instantiation process can be partly automated, for example the direct

concept mappings can be extracted from the ontologies and then fed into specific fields in the view templates.

View templates are also defined with concepts in the semantic model. Figure 26 is the ontological bindings for the view template observation, defining each data row in the virtual table called ViewObservation must have the following fields: ObservationValue, DeterminandID, MediumID, UnitID, AnalyticalFractionID, Time, and StationID.

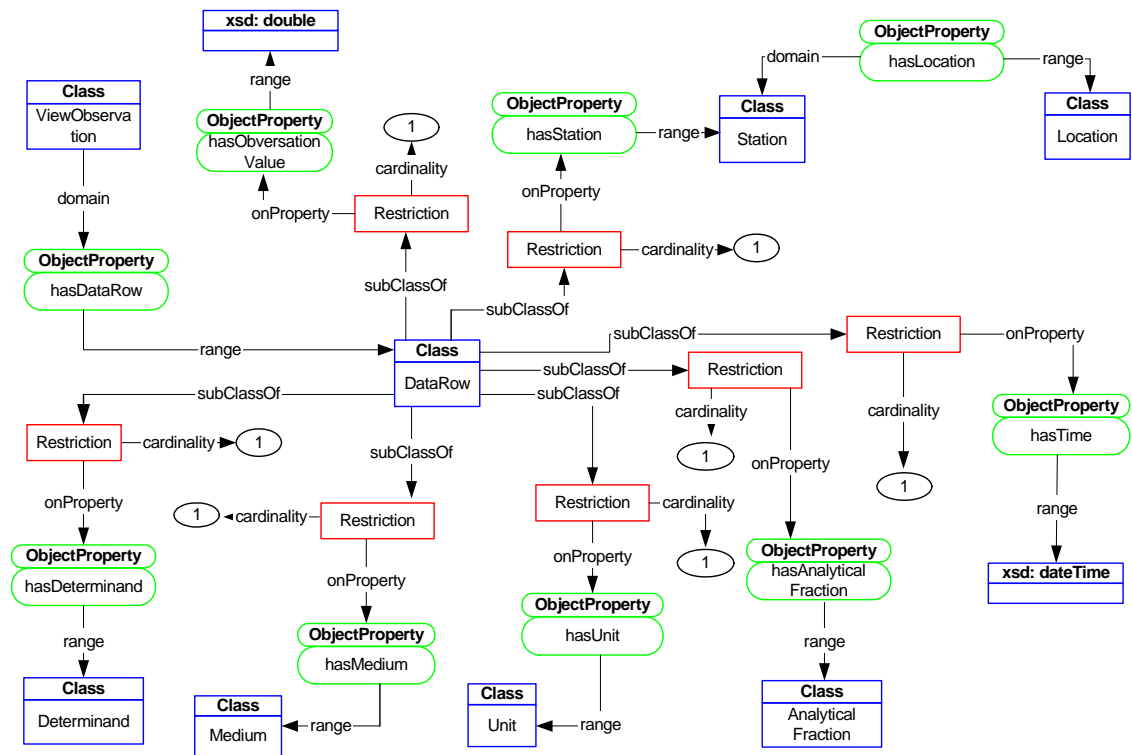


Figure 26 Ontological bindings for the observation View template

5.3.6 Constraint Modelling

Rules are used to model some constraints. For instance, regions are made up of stations within a certain range, in other words regions can be seen as constraints on the stations' locations. Adopting a simplified division, a country has four regions, North-East, North-West, South-East and South-West. A central point with coordinates (x_c, y_c) is calculated from the formulas $x_c = (x_{max} + x_{min}) / 2$ and

$y_c = (y_{max} + y_{min}) / 2$. The North-East region is defined as: any location with coordinates (x, y) where $x > x_c$ and $y > y_c$. Other regions are defined in the same way.

```

<ruleml: imp>
  <ruleml: _rlab ruleml: href="#regionCalculation"/>
  <owlx: Annotation>
    <owlx: Documentation>The region of a station depends on the coordinates
    relative to a central point</owlx: Documentation>
  </owlx: Annotation>
  <ruleml: _body>
    <swrlx: individualPropertyAtom swrlx: property="&ex; #hasLocation">
      <ruleml: var>station</ruleml: var>
      <owlx: Individual owlx: name="&ex; #Locality"/>
    </swrlx: individualPropertyAtom>
    <swrlx: datavaluedPropertyAtom swrlx: property="&ex; #hasCoordinateX">
      <ruleml: var>station</ruleml: var>
      <ruleml: var>coordinateX</ruleml: var>
    </swrlx: datavaluedPropertyAtom>
    <swrlx: datavaluedPropertyAtom swrlx: property="&ex; #hasCoordinateY">
      <ruleml: var>station</ruleml: var>
      <ruleml: var>coordinateY</ruleml: var>
    </swrlx: datavaluedPropertyAtom>
    <swrlx: builtinAtom swrlx: builtin="&swrlb; #greaterThanOrEqualTo">
      <ruleml: var>coordinateX</ruleml: var>
      <owlx: DataValue owlx: datatype="&xsd; #int">centralX</owlx: DataValue>
    </swrlx: builtinAtom>
    <swrlx: builtinAtom swrlx: builtin="&swrlb; #greaterThanOrEqualTo">
      <ruleml: var>coordinateY</ruleml: var>
      <owlx: DataValue owlx: datatype="&xsd; #int">centralY</owlx: DataValue>
    </swrlx: builtinAtom>
  </ruleml: _body>
  <ruleml: _head>
    <swrlx: datavaluedPropertyAtom swrlx: property="&ex; #hasRegion">
      <ruleml: var>station</ruleml: var>
      <owlx: DataValue
      owlx: datatype="&xsd; #string">NorthEast</owlx: DataValue>
    </swrlx: datavaluedPropertyAtom>
  </ruleml: _head>
</ruleml: imp>

```

Figure 27 Constraint of region "north-east" modelled using OWL-S

The rules are implemented using the RuleML support in OWL. An example of the rule defining the “North-East” region is shown in Figure 27.

5.3.7 Process for View Instantiation and Example

Figure 28 shows the whole process of query decomposition, keyword extraction, view template instantiation and finally SQL generation.

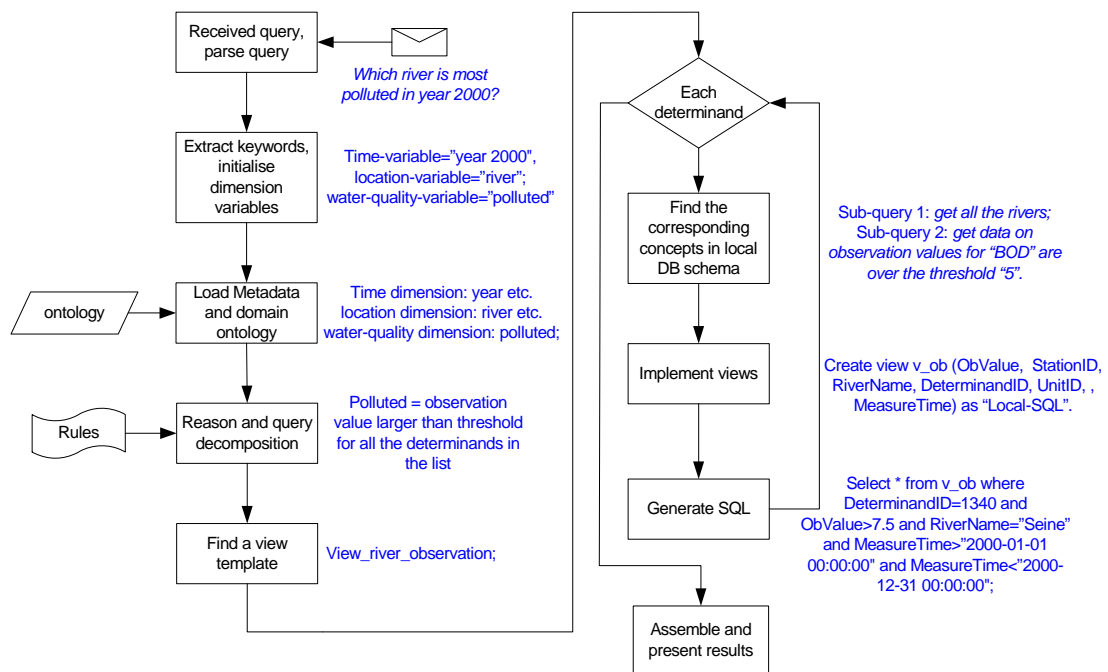


Figure 28 Workflow for SQL generation using views

The instantiated SQL command to generate river observation view based on IOW schema is as follows: “*CREATE VIEW view_river_observation (ObValue, StationID, RiverName, DeterminandID, MeasureTime) AS SELECT resultat_analyse, mesures.code_station, troncons_hydrographiques.nom_cours_eau, mesures.code_parametre, date_operation_prelev FROM mesures, stations, troncons_hydrographiques WHERE stations.code_station=mесures.code_station AND stations.code_hydro=troncons_hydrographiques.code_hydro;*”.

5.4 Evaluation of the Framework

5.4.1 Descriptions of the Experiments

The experiments to evaluate the framework focus on two aspects: the use of service quality data in improving the service discovery effectiveness; and the use of the semantic virtual data warehouse approach with specific DSS queries.

The objective of the first experiment is to verify that the service discovery mechanism is enhanced by incorporating service quality data into the match-making process. The first experiment compares the utilities of two users, one that matches the service capability only, and the other that matches service quality as well as service capability. For the second experiment, the objective is to demonstrate the functions of the semantic virtual data warehouse. The experiment is based on testing illustrative queries such as “which river is the most polluted in year 2000 in all countries” and comparing the expected against the actual results.

5.4.2 Comparison of Different Service Discovery Methods

The main criterion to compare different service discovery methods is how it increases the service requester’s utility. This experiment is conducted to compare the utilities between two match-making methods for service discovery: one method matches the service capabilities only, the other matches both capability and quality. More specifically, the experiment is implemented as a simulation, with 12 service providers and 2 service users. Among the 12 providers, half of them have one type of service capability, while the other half has another type. All of them however have different efficiency parameters (that are randomly allocated by the simulation). The first user adopts the capability only matching method, and the second one uses the method that matches both capability and efficiency. Here the service providers are assumed to have the same degree of robustness; therefore the capability and efficiency are given equal weight in the utility calculation. The results in Figure 29 shows that generally user 2 has a higher utility than user 1, which can be interpreted that the second matching method is more effective than the first method.

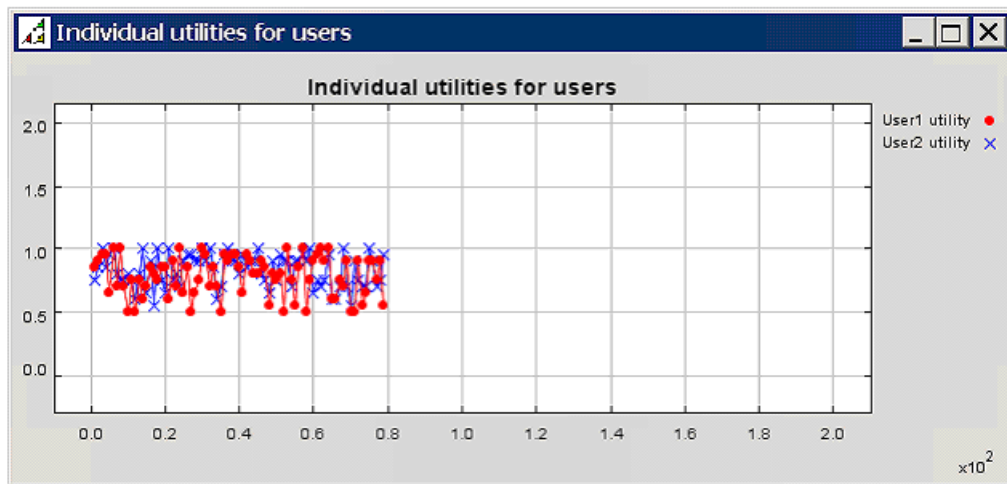


Figure 29 User utility measurements of different service discovery mechanisms

5.4.3 Data Integration and Harmonisation using Semantic Views

An example SQL query to retrieve observation data on Nitrate over threshold 7.5 at river SEINE during year 2000, using the river observation view is: *select * from view_river_observation where RiverName="SEINE" and DeterminandID=1340 and ObValue>7.5 and MeasureTime>"2000-01-01 00:00:00" and MeasureTime<"2000-12-31 00:00:00";*. The result of this query is plotted in Figure 30.

Data results for Q1 “*which river is the most polluted in year 2000*” produce observation values for certain determinands at different rivers. These results then serve as raw data to provide the final ranking of polluted rivers to the end user. Several ranking criteria exist for the more polluted the river, e.g., based upon the number of measuring stations with bad water quality or based upon the overall average of all the observations over the stations with bad water quality determines the most polluted river.

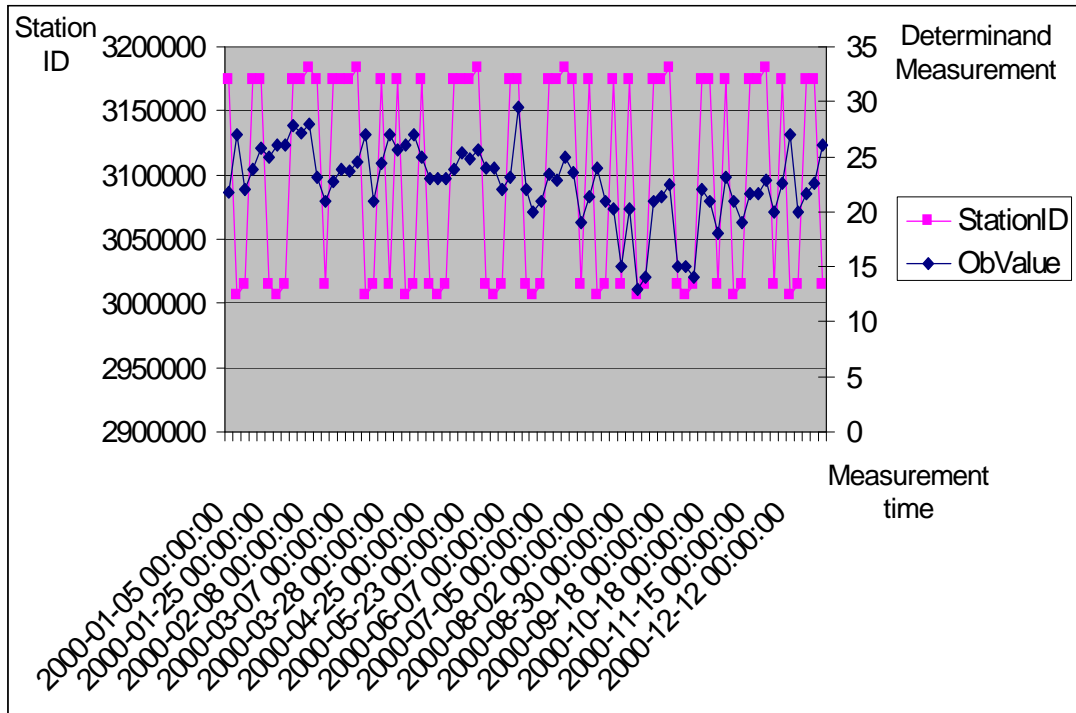


Figure 30 Graph of observation values for Nitrate in river SEINE during year 2000

Here observation values for Nitrate at three different rivers namely SEINE, BLAVET, and AUBE are plotted in Figure 31.

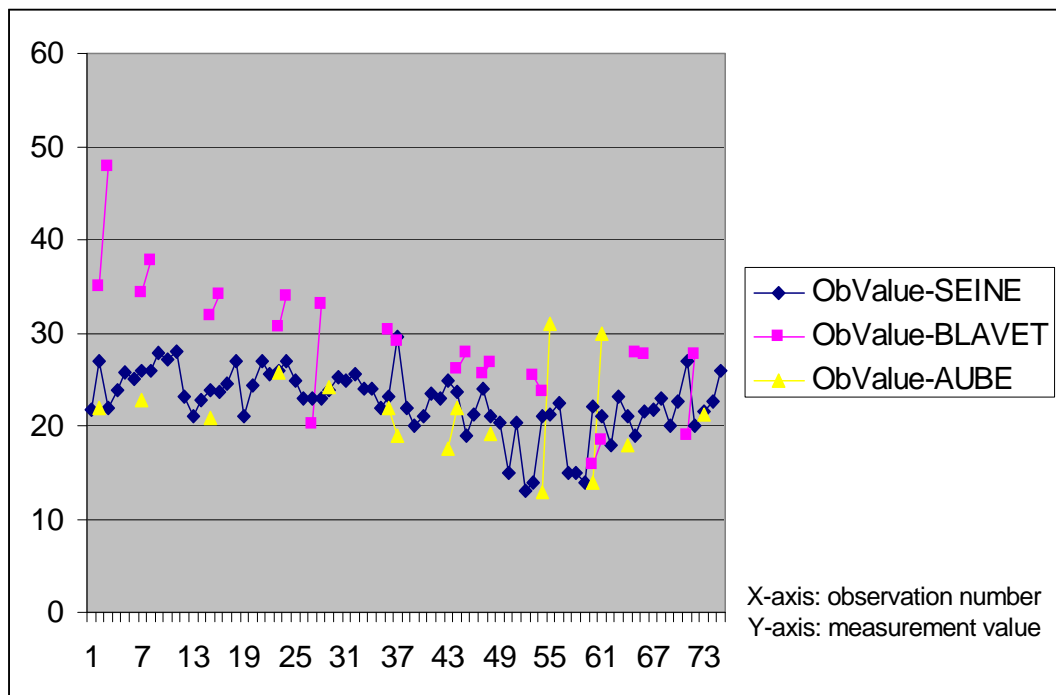


Figure 31 Observation values across three rivers

The same set of data has different readings according to different ranking criteria, which is shown in Table 15.

Table 15 Different ranking results

Ranking methods	Result (descending order)
Most polluted stations first	Seine (5), BLAVET (2), AUBE (1)
Highest average measurement value first	AUBE (40.56) BLAVET (28.73) Seine (24.23)

5.5 Discussion

The architectural overview in Figure 16 shows that three new agents (MDDVA, MA and SQA) have been added into the framework as well as an application ontology called MDDVO, comparing to the overview in Figure 7 in Chapter four. The MDDV Agent is dedicated for high level data analysis based on multi-dimensional data view structures. Both MA and SQA are more concerned with metadata management than directly related to query processing. In terms of system functionality enhancement, there are two aspects. Metadata available from MA and SQA helps the directory to be more precise in service discovery, that it matches service capabilities as well as qualities. The distributed structure of the metadata model also eliminates the dependency on a central directory. The metadata management and directory requirement outlined in section 2.1.4 is considered fulfilled. Another enhancement is that the semantic virtual data warehouse model provides a more flexible solution for multi-dimensional data view and analysis. This fulfils the derived data analysis requirement.

Agents are enhanced with a local metadata repository, which can be seen as a primitive version of agent memory. The local metadata repository mainly stores query and interaction metadata, and provides the access and necessary processing for the metadata. The metadata may be sufficient for local service discovery, i.e. the service provider is selected from previous interaction records, thus providing an alternative to consulting centralised repository for service discovery. The interaction

metadata can also be processed to retrieve service quality information, for example a query initiator can deduce the latency of the query from the time when the reply is received and the time when the query is sent. With the monitoring and storage of metadata, agents are no longer state-less. This is a significant advance compared to those in the chapter four framework. At the system level, having interaction metadata enables provenance checking. There is a clear record to refer to in case of exception and service interruption.

The service discovery mechanism proposed here enhances both the availability and effectiveness comparing to conventional service-oriented centralised approach. In particular, adding distributed local metadata repositories and alternative service discovery routes into the system no doubt eliminates the dependency on a centralised directory, i.e. improving the availability of service discovery. Also more comprehensive metadata model on service providers that include service capability and quality enhances the effectiveness of service discovery. However a hybrid directory structure with distributed local copies in agents means consistency is a major management concern. Having to actively collect service quality measurements also adds processing overhead into the system. These are however the price to pay for more available directory service and more efficient service discovery.

The proposed virtual data warehouse using semantic views advances the DB-based information integration approach in several aspects. Firstly, application semantics is also modelled in an ontology, and is linked to other domain knowledge, which exploits the relationships between application concepts and domain concepts to relate data views to the domain conceptualisation. Secondly, views can be changed so easily without minimal impact in the actual data storage to accommodate new application analysis requirements. Thirdly, the creation of views can be partly automated using direct mapping between concepts in application and domain ontologies.

However the framework is not without its limitations. One of the limitations is that the lack of functions to deal with data corruption and missing data. The reason why data corruption is not detected in the system is that to detect the incorrectness of the data value requires in-depth knowledge of the domain. As to the missing data, it is

difficult to detect because measurements of water quality are taken at irregular intervals, meaning it can happen that one station monitors the water quality every hour while another station only monitors this once a day. The direct consequence of the irregularity is that detecting what data is missing requires detail specifications on how the data is measured. Information at this level of detail is not necessarily available for the application to detect missing data.

5.6 Summary

This chapter focuses on adding flexibility into the EDEN-IW system to cope with open service environment, specifically on the following aspects: agent interaction to select receivers, service discovery and derived data analysis. The description of the framework is in three parts: the overview of the framework with design issues, decentralised directory structure, and semantic multiple dimensional data view. The proposed framework enhances agents' service selection using a local metadata repository and use richer communication patterns and QoS metadata to enhance service selection. It also supports semantic based virtual data warehouse with views. Evaluations are conducted in particular to verify the effectiveness of the enhanced service discovery and data harmonisation using semantic views.

6 EDEN-IW MAS Extension 2: Fault-Tolerance

6.1 Introduction

The robustness of services has not been addressed in the MAS framework given in chapter four and chapter five. These assume that the framework does not fail. Maintaining the normal system operation in the face of faults, fault-tolerance, remains a key management concern. Distributed computing systems especially those with heterogeneous components are more prone to failures caused by interoperability problems. Fault tolerance can in turn improve service responsiveness and availability. The system presented in chapter five improves the availability of the directory service by distributing metadata into individual agents and interlinking metadata to search for service. This decentralised approach also improves the flexibility of service discovery by enabling alternative service discovery paths.

The fault tolerant framework aims to support graceful degradation of services so that the system remains partially operational in the presence of partial exceptions instead of coming to a complete halt. The concept of MAS system or collective management goals is used to differentiate the tasks that are crucial to the MAS operation based upon benevolent behaviour of its component parts and those that are not. All process tasks that directly contribute to the achievement of the MAS goals are considered to core tasks (to achieve the MAS goals). Sub (sub-optimal) goal states are worth considering if the original goal is no longer achievable. Hence, goal states can be further divided into sub-goals and the tasks that cause the MAS to shift state from the start and current state towards the goal state and modelled as a Hierarchical Task Network (HTN) [95]. Agents are responsible for achieving their individual sub-goals, which collectively contribute to the system goal. For an information integration system, the ultimate goal is to provide users with complete data in the most effective manner. Sub-optimal states involve compromises on either the completeness of the data set or the time constraint. In other words, if the quickest path to the full set of data is not available, try the next quickest path; and if the full set of data is not available, reply with partial data. These sub-optimal states help define the exception handling strategies needed for specific situations.

The exception handling service needs to identify exceptions and isolate corresponding process or agent to prevent failure propagation. The categorisation and effects of the exceptions is detailed in section 6.1.2. Exceptions are categorised by the effects they cause, so that appropriate handling measures can be specially devised and implemented. It is not feasible within the constraint of this research project to design a generic fault tolerance model so comprehensive that every single exception is handled.

Two handling strategies are proposed, one for total agent failures and one for partial agent failures. The one for total agent failure uses a substitute, a new instance of the same type of agent, which takes over the activities of the failed agent. The interaction metadata helps the substitute agent to be restored to the last state before the original agent fails and continues with the conversation. As to the strategy for partial agent failures, it makes use of the exception context and a semantic model of agents' dependencies in order to identify the most appropriate handling measure.

The rest of this chapter is structured as follows. The introduction continues with the problem description, and an analysis of exceptions and their effects. The characteristics of the fault tolerance model are then outlined based on the targeted exceptions. Design issues follow to discuss several key problems, such as the modelling of heuristic knowledge, and the necessity of hypothesis testing. An overview of the framework provides details of the various exception handling components including the monitor function of agents, exception detection, the semantic model for exceptions and processes and the fault handler selection. Some test cases are described together with experimental results to evaluate the exception handling model.

6.1.1 Problem Description

The distributed information system described in chapter five is complex, in that not only there are many interconnecting components, each of which is heterogeneous, but also there are many levels of communication between these components such as inter-agent communication, non-agent service invocation, and infrastructure level message transmission. Each of these components is subject to failure, e.g., the

communication link is not always stable; conflicts may occur in accessing the same resource. The query processing workflow of the MAS depicted in chapter five does not take into account the exceptions that are caused by the environment not being fully observed, the environment may change during system operation. Figure 32 shows several potential exceptions during query formation and during the analysis stages of the workflow. This example illustrates some typical exceptions and the branching of the workflow to handle exceptions.

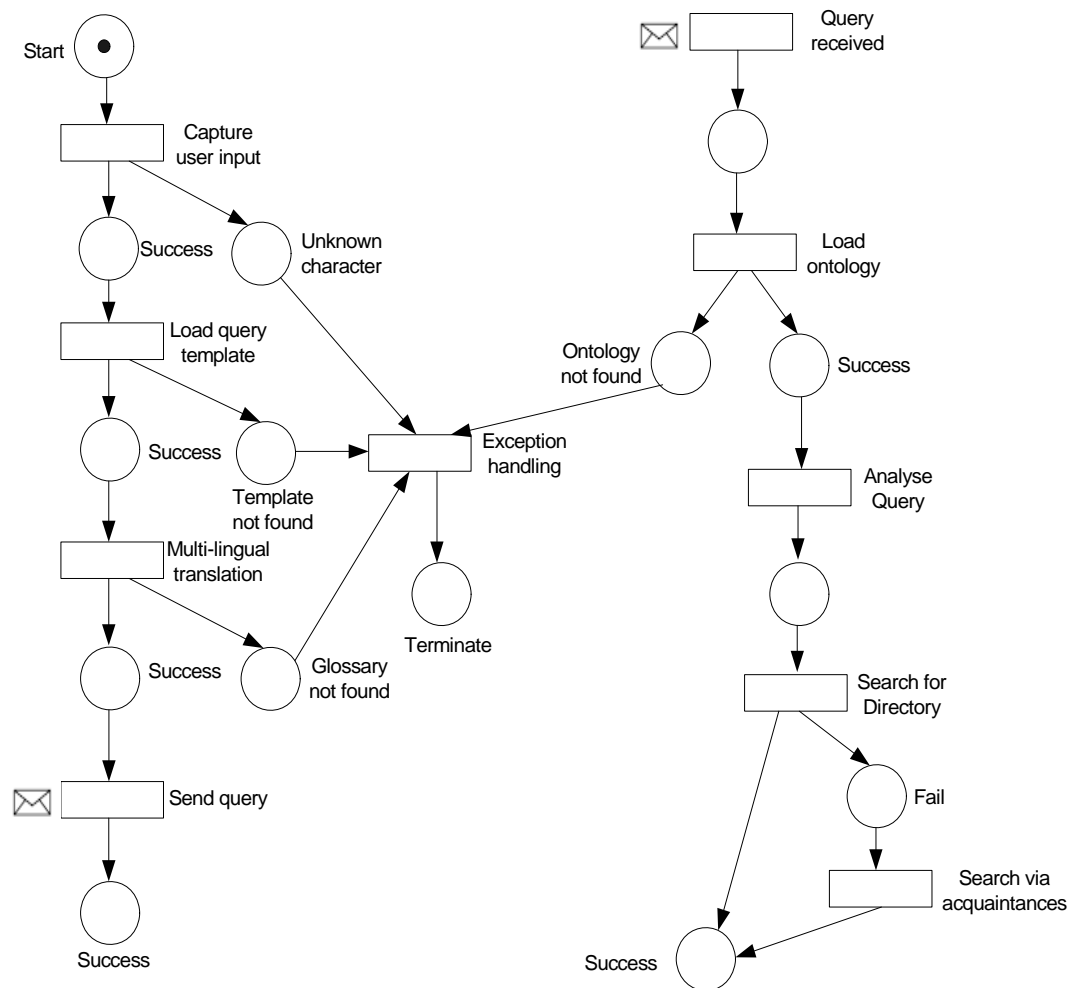


Figure 32 Query forming and analysis workflow with potential exceptions modelled in Petri Net [88]

Example exceptions include: ontology files fail to be loaded; ontology files have been updated but not the corresponding parser; the multi-lingual glossary cannot be found; a SQL query fails because the generated SQL query has the wrong syntax for

the RDBMS and high level user queries cannot be mapped to low level queries to the database.

At the individual agent level, the process workflow is complex as shown in Figure 33.

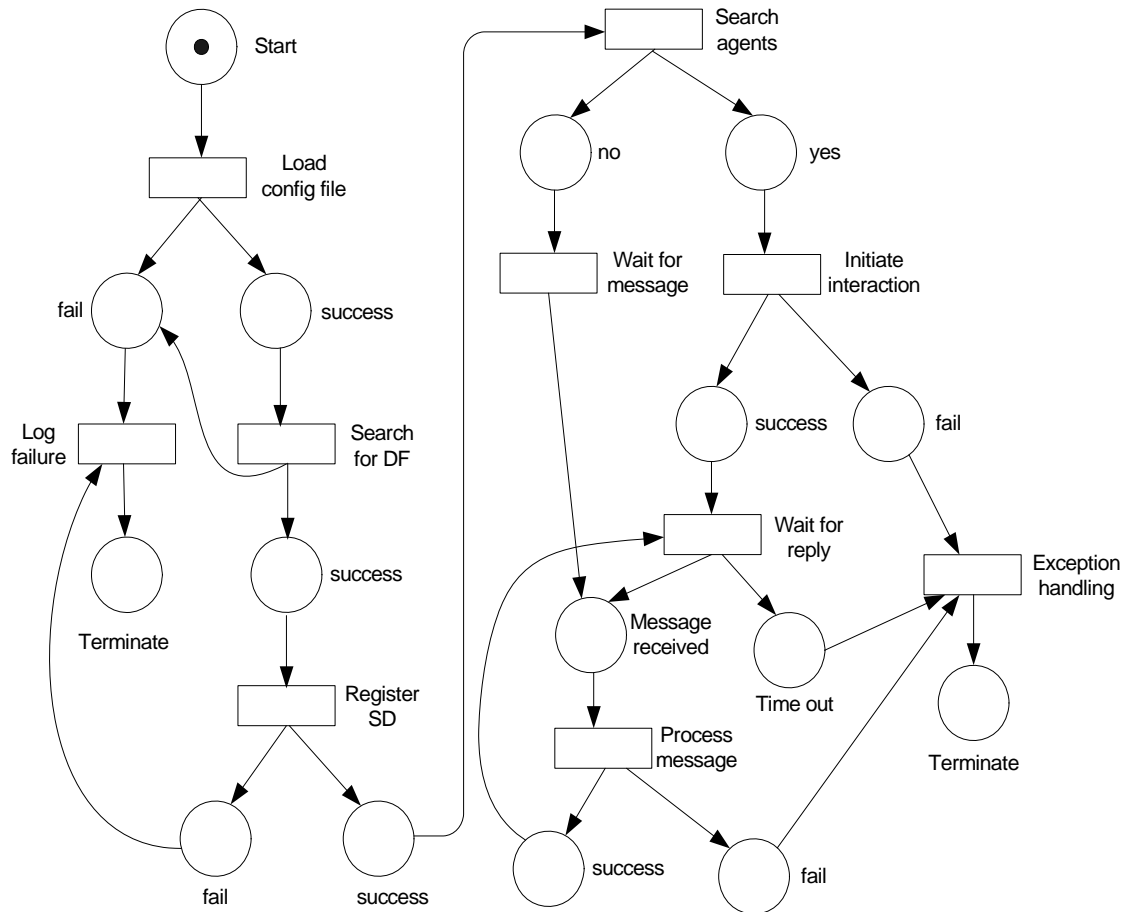


Figure 33 Agent's query processing workflow with exceptions in Petri Net [88]

6.1.2 Exceptions and Failures

It is not realistic to devise a system that is able to handle every potential exception in real time, simply because some exceptions require human intervention, and also because not every exception can be foreseen at the design stage. Also there are many types of categorisation of exceptions with different view points. This can make reaching a consensus amongst the different stake-holders difficult. It is proposed that exceptions are simply categorised into total agent failures and partial agent failures.

The effects these exceptions cause in turn can be categorised into recoverable, degradable, and hard failures.

The sub-set of exceptions being handled in this framework are those that cause recoverable or degradable failures, because the handling process can be fully or partially automated. More specifically, recoverable failures include: total agent failure, assuming agent functions are replicable, and function failures where alternative paths exist to achieve the same goal. Degradable failures refer to those that cause incompleteness in data results. For instance a query is decomposed and distributed into multiple different data sources, and one of the resources (agents) fails and cannot provide a reply to a query, therefore only partial data results are replied to the requester.

As to most cases of hard failure, the system shuts down and human expertise is required to debug and modify the software or hardware. However some precautions are taken to minimise the occurrence of hard failures in the system. Some typical exceptions that lead to hard failures are examined, for instances Null Pointer Exception and File Not Found Exception are two of the commonly occurring ones. Therefore simple catch mechanisms can be used to report these exceptions before they cause hard failures, e.g., catching Null Pointer Exception by checking whether or not the variable value is null before using it.

A fault is defined as the cause of an exception. Fault diagnosis is an important issue in fault tolerance techniques, because in some cases if the cause of the exception is identified, it helps the handling process to be more effective, i.e. curing the cause rather than just the symptoms. This eliminates the reoccurrence of the exception. Some researchers even distinguish causes in terms of being active or inactive. Inactive faults may not cause detectable exceptions in the system. Here only active faults are considered, because if a fault does not cause a system to malfunction, then it is not a priority to be handled.

However identifying what causes an exception to occur can be very difficult to determine especially in a dynamic environment where situations may be transient. In addition, some exceptions may have several causes, and one cause may generate

several exceptions. Hence, it is not a simple one-to-one mapping from exception to its cause, it is often a many-to-many mapping problem. Fault diagnosis is essentially a reasoning process that incorporates the exception context to identify the most probable cause of an exception. Note that what is generated by the end of the diagnosis is only a hypothesis, not a validated cause. The hypothesis then needs to be verified. It is clear that fault diagnosis is a costly process, and it is not always the most effective way to perform exception handling. For example in relation to hard failures, it is almost not worth the extra effort to identify the fault because any faults that may occur often require human intervention to handle them.

6.1.3 Architectural Overview

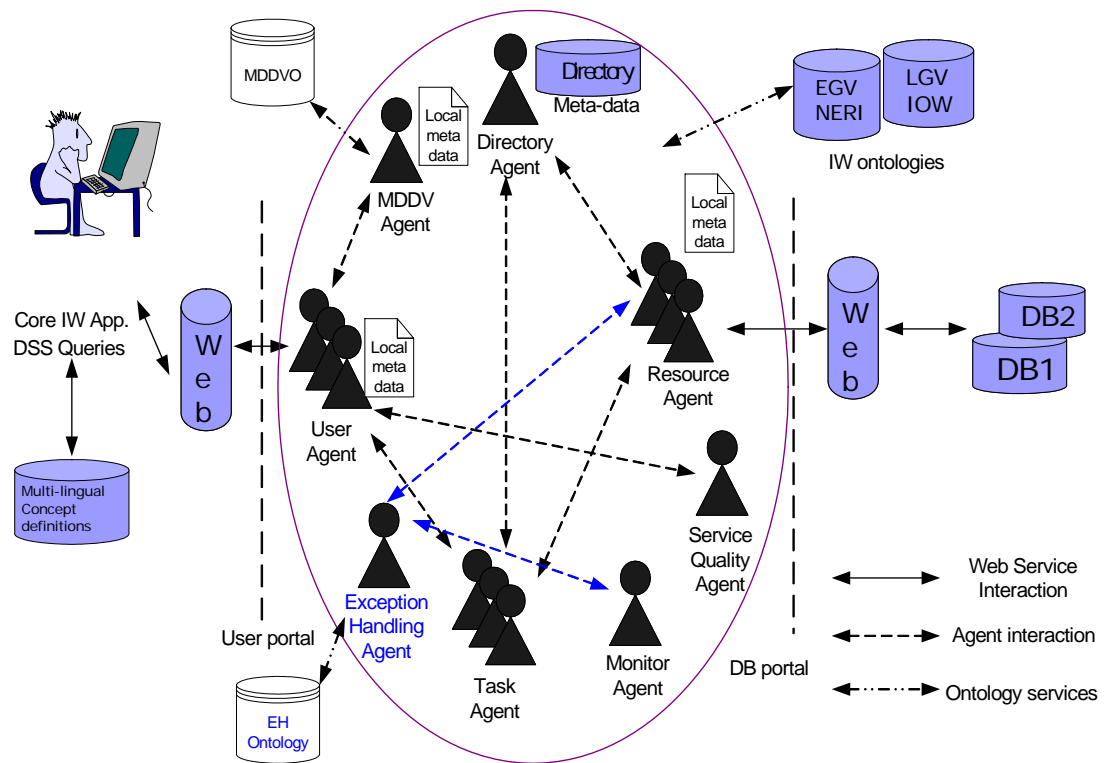


Figure 34 Architectural overview with added exception handling agent and exception handling ontology

The Exception Handling Agent has been added into the system with an exception handling ontology, as can be seen in Figure 34. Note that this is an illustration of the system architecture not a detailed design model of the whole interaction, so although it only shows two agent interactions with the EHA, from Resource Agent and

Monitor Agent, it does not mean other agents do not communicate with EHA. On the contrary, any agent in the system will contact EHA, when exceptions have been detected by that agent.

The Exception Handling Agent or EHA communicates with other agents in the system using the FIPA ACL, which means that there is no need to design a dedicated service action interface for the fault-tolerance model, it uses a generic one represented in RDF. The added exception ontology to define the specific application terms for this is modelled using the OWL language. The underlying messaging and networking functions used are the same ones in the system described in chapter four and five.

6.2 Characteristics of a Robust MAS in Handling Information Queries

A robust MAS for handling information queries should support high availability and responsiveness. Availability refers to the fact that the service persists even when exceptions are present. Responsiveness concerns reducing non-deterministic delays the system suffers and dealing with the various uncertainties involved in the query answering process.

Two major objectives of the EH model are to improve both service availability and responsiveness. Availability can be improved by reducing single point of failure, by having alternative paths to the same goal and by using substitutes. Responsiveness can be maintained by discarding long over-due replies or by actively making use of a time-out mechanism to avoid long response times.

Availability is also a key factor that measures the effectiveness of fault tolerance techniques, and the more available the services are the more fault tolerant the system is. These two objectives consequently determine how the exception handling model is designed.

6.3 Design Issues

6.3.1 Modelling of Goal Tasks and Plans

Traditional AI defines planning in terms of producing a sequence of actions that will achieve a goal [95]. The plan is the outcome of the planning process. More specifically, adopting the definitions from HTN, planning problems are described using three types of tasks: goal tasks, compound tasks, and primitive tasks. A goal task is a task of satisfying a condition. A compound task is a complex task composed of a sequence of actions. A primitive task is an action that can be executed. A plan can be seen as a full or partial order of primitive tasks that achieve the goal task. Goal tasks for agents are defined by the services they provide and the Agent Interaction Protocols they support. This is also to say the services are an agent's goal tasks.

For example, the “Distribute-Query-Service” supported by the Task Agent can be seen as a goal task that in turn consists of the following compound tasks: service discovery (metadata query), distribution of queries, and collection of replies. Service discovery is a compound task that can be achieved with three different plans: querying local metadata repository, querying the directory, and broadcast for the service. The decomposition of goal task is illustrated in Figure 35.

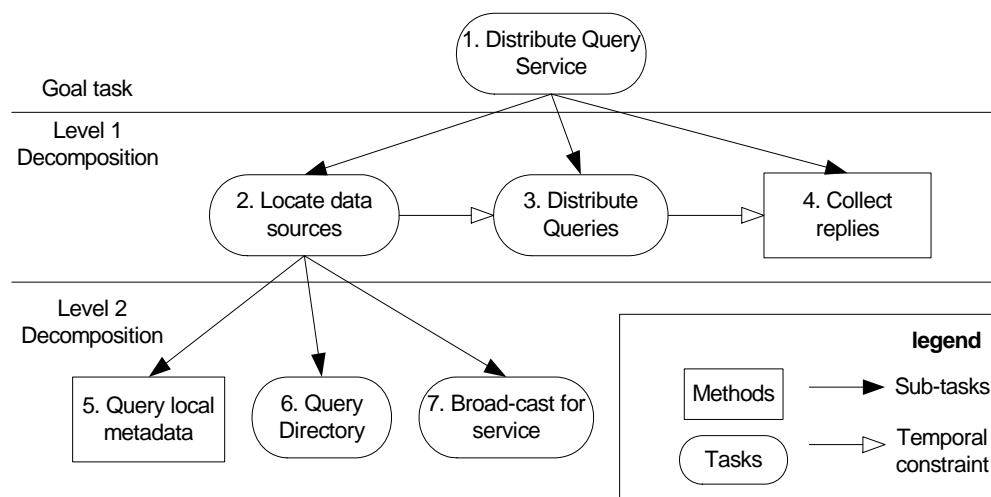


Figure 35 An example of task decomposition in HTN

Plan generation is the process of decomposing goal tasks into primitive tasks with a specific order. For the previous example, a level 1 plan consists of three tasks:

service discovery, distribute queries, and collect replies. There are three level 2 plans for service discovery: query directory, query local metadata repository, and broadcast to identify a service. The next level plans are very detailed and involve program constructs. For example to query directory, the preconditions are that the directory agent is available and the address known, so that the level 3 plan would look like: IF directory agent available and address known, THEN send metadata query to directory.

Tasks on level 1 are defined in the agent's service descriptions and the supported AIP transition model. Tasks on level 2 are defined in the plan library. Plan library consists of plans that could achieve the same goal, or plans that achieve sub-optimal goals. Having alternative plans that achieve the same goal is one crucial way of improving availability of the goal because single points of failure are avoided.

Re-planning may also be necessary in case the original plan is no longer valid due to changing circumstances. Re-planning assesses the current situation to see whether the original goal is still achievable and to decide alternative plans, accordingly. Re-planning is an important part of a fault tolerant framework.

6.3.2 Finite State Transition Models for Agent Interaction Protocols

A safe Agent Interaction Protocol (AIP) specifies not only normal states but also exception states during the course of the agent conversation. Failures shown in Figure 36 have not been individually distinguished because a failure state may be triggered by several kinds of exceptions. For example the failure state from "send message" may be caused by the message having a wrong receiver address, or the underlying network failure. The individual exception event is not further distinguished in the transition model because it is the fact that the failure state has been reached, and the subsequent transitions from the failure state are more important to exception handling. Also the transitions shown in this figure is a general illustration of message exchange which is not specific to any AIP.

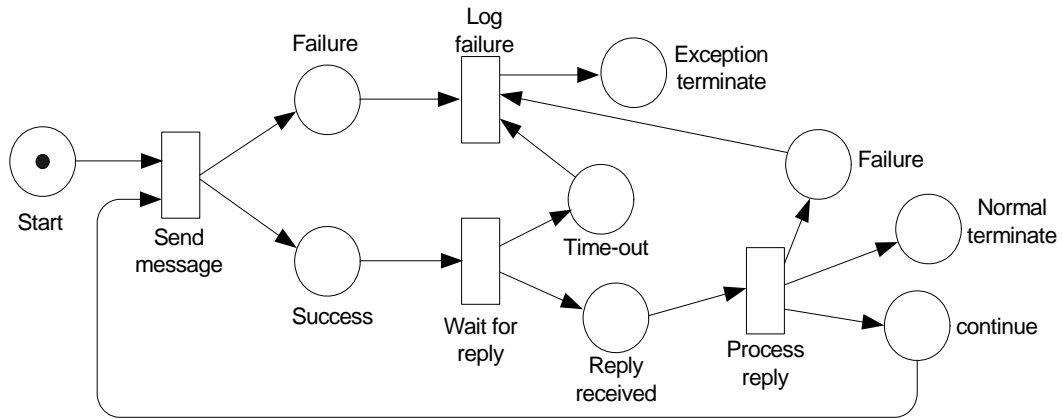


Figure 36 Message processing transition model in Petri Net

The transition model specific for one AIP is split into several sub-models according to participants' roles. That is to say a FIPA-Request is split into two sub-models because there are two roles of participants, the request initiator and the responder. There are four variations of the reply being modelled in the FIPA-Request protocol, agree, refuse, failure and inform, as shown in Figure 37. Consider the response to a request, its state transition model is given in Figure 38.

The AIPs which an agent supports together with the role it plays are defined in an agent's service description. These AIPs are then translated from the transition model into state-transition-mapping tables. These mapping tables define the states, transitions, conditions and next states. They play a key part in the planning and re-planning process.

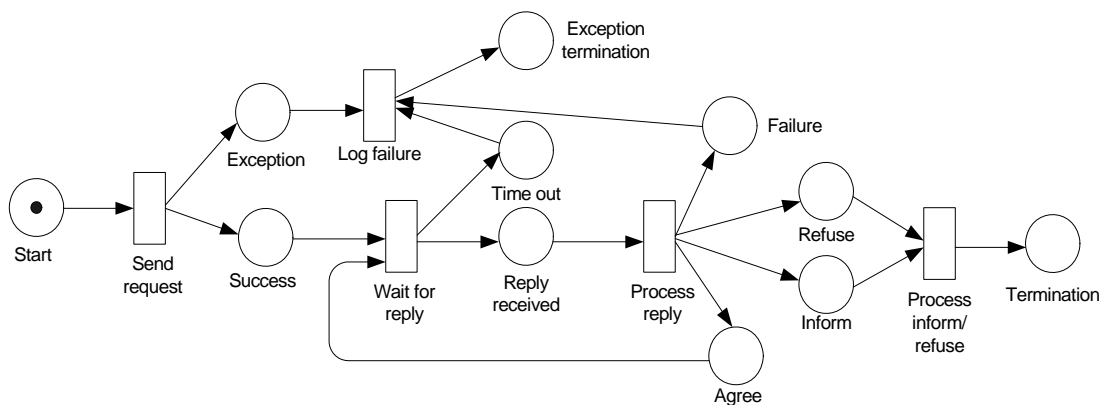


Figure 37 State transition model for FIPA-Request initiator in Petri Net

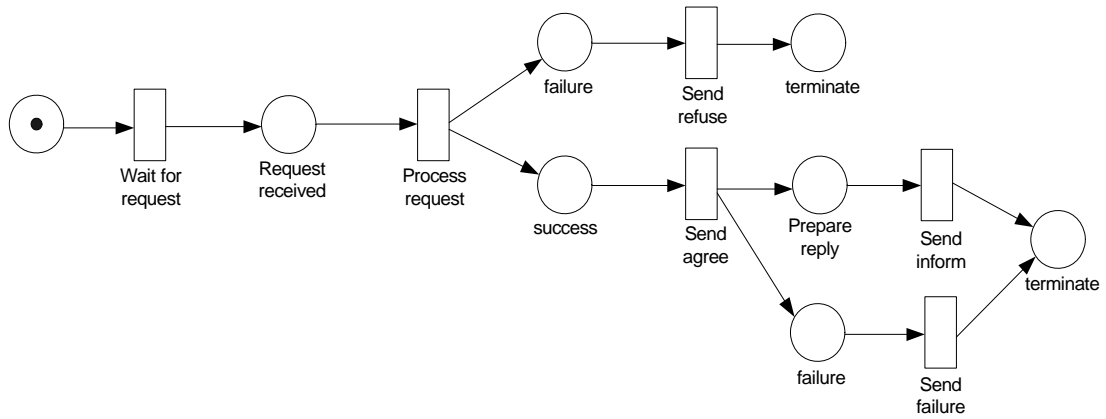


Figure 38 State transition model for FIPA-Request responder in Petri Net

6.3.3 Exception Detection & Fault Diagnosis

The detection process is different for partial agent failure in contrast to total agent failure. The former is detected by the monitoring unit embedded into each agent that detects the occurrence of abnormal situations. The latter is detected at the Monitor Agent by checking the consistency of agent interaction patterns and detecting any anomalies at the system level.

Each agent uses a monitoring unit that dynamically monitors the functioning of agent processes to differentiate abnormal situations from normal ones. This works as follows. Agent processes can be categorised into internal activities or external interaction with either another agent or a non-agent service. The correct operation of the internal activities is partly achieved by careful design and implementation. By checking the output these processes generate against the expected output, the monitoring unit can decide whether the processes are functioning normally or not. For instance, agents have an internal process for generating ACL messages with given parameters. If the generated message is checked and is malformed then the monitoring unit assumes that the message generating process is faulty. As to the external interactions, the exception states can be inferred from the state transition tables of the normal AIP or from the non-agent service specifications. Whenever messages are received in the agent, they are parsed and mapped to specific states in the transition tables, and if the state is an exception state, then the exception is reported.

Total agent failures are detected at the system level, because each agent is designed to send the agent platform directory facilitator a “deregister” message when it dies. Agent death is used here to simulate the natural disappearance of an agent, and it is not a result of poor design. Upon receiving the “deregister” message, the DF reports a total agent failure to the EH, which starts polling the interaction metadata from the MA, which keeps a record of the agent interaction in the system, and initiates the appropriate error handling processes.

To identify potential faults for an exception, human experts’ knowledge of the system as well as the exception context is crucial. Because the fault diagnosis is a costly process and because not all exceptions can be attributed to exact causes, it is decided that only selected degradable failures are diagnosed for faults. For example, when TA receives an empty-result reply from RA whereas normally the RA *should* have data for this query according to information received from the DA, a no-data situation triggers exception handling. The conflict between the metadata query result and data query result indicates two possible explanations, either the metadata is stale or something is wrong with the data query process. More specifically, it could be caused by the update in the data source not being propagated to the metadata repository, or a case where resource agent has some internal error in loading an appropriate ontology, hence providing empty result. This example shows that if the no-data situation has not been identified as an exception, the cause would not be identified and treated, and the exception persists.

6.4 The Exception Handling (EH) Framework

The EH framework is described in more detail here, starting with the EH Agent and the exception handling workflow description, followed by the explanation in detail of how the monitoring and re-planning are used by the individual agents. Then the semantic service description is clarified, showing how the exception handling process is modelled as part of the semantic description for a process sequence. Heuristic knowledge is represented as rules that form the core of the exception handling framework deciding which handling strategy to take. Finally the two main methods to handle total agent failures and partial degradable failures are given in section 6.4.5.

6.4.1 EH Agent and Exception Handling Workflow

The Exception Handling Agent or EHA is created to handle exceptions in the system by assessing the overall situation, deciding the appropriate strategy, and giving instructions to coordinate agent interactions. It is essentially, a centralised EH service similar to Klein's [66], but it is made an agent so that it speaks the same language with other agents, eliminating the need for a dedicated EH language as in the Klein model. The monitoring unit in each agent practically acts as sentinels, similar to Haegg's approach [50], yet the monitoring units are part of an agent and communicate with EHA instead of to each other.

A general exception handling process works as illustrated in Figure 39.

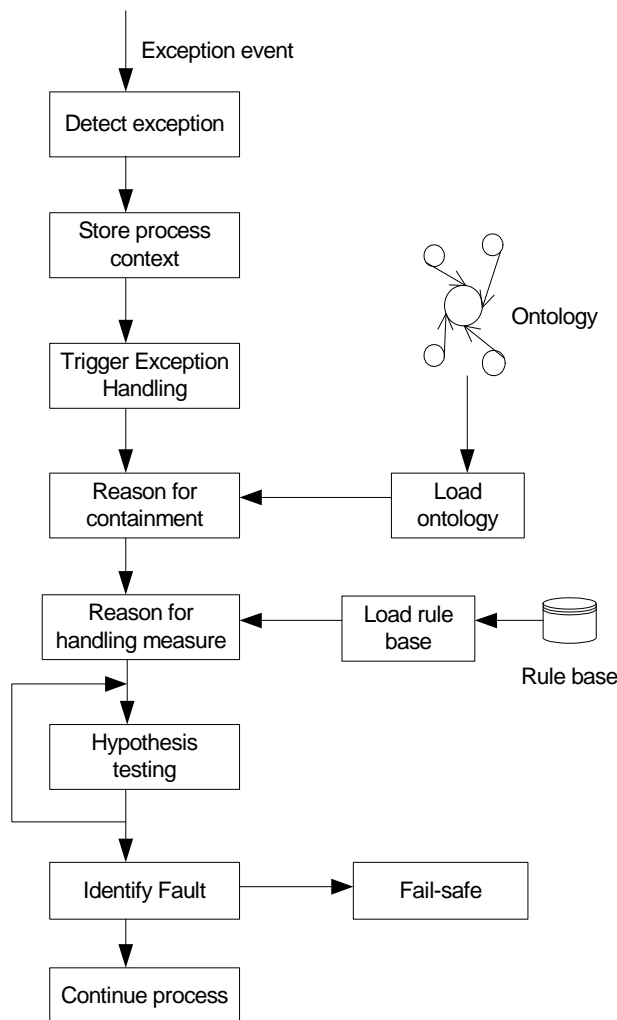


Figure 39 Generic exception handling workflow

When an exception event is detected, either from an agent's monitoring unit for partial failures or the agent platform DF for detecting total agent failures, handlers are triggered to report the exception, together with the process context to the EH Agent (EHA). The EHA then collects other information about the event if necessary, and loads the rule base to identify the most appropriate handling method.

It is optional whether or not the EH model decides to test any possible hypothesis generated during the fault diagnosis process, to pinpoint a fault. It is worth noting that hypothesis testing might reiterate, therefore it can be time-consuming, which again may affect the agent's utility. There is a trade-off here to whether or not to make an extra effort in pinpointing the cause of the failure or to mask it and resume the remaining services.

6.4.2 Monitor and Re-planning Functions for Agents

Agents have a planning unit and an execution unit to generate and carry out sequences of actions towards fulfilling their goals. While executing the actions, agents also monitor any abnormal behaviours or situations as they occur. If any discrepancies arise, the monitor unit reports to the planning unit so it can re-plan to bypass the exception, as well as flags an exception to the exception handling agent.

Basically the re-planning consists of the following steps: evaluate if the original goal is still valid, i.e. that is achievable and has the highest priority; identify a new goal if the original one is invalid; evaluate if the original plan can achieve the original goal or generate a new plan for the new goal. The decision on whether or not to change the original goal partly lies in how serious the exception is and if there is an alternative that can bypass the exception.

Therefore the plan library plays an important role in re-planning, because it stores alternative plans that achieve the same goal, or plans to achieve sub-optimal goals. Concerning the two plans for service discovery, i.e. query directory service or broadcast for the service, they are equal in that they both are able to achieve the same goal but one might be more efficient than the other. In cases where the query efficiency is crucial, the more efficient plan will be assigned a higher priority in the

re-planning. Of course if the time constraint of the query is not an issue, then they will be considered equal and have the same chance of being chosen.

The re-planning has to take into account the input from EHA when coordination between agents is needed to mask the exception. For example, when a resource agent detects its link to the data source has broken down, it first of all evaluates the failure this exception would lead to. This exception stops the communication between RA and DB completely, therefore none of the RA functions that require data retrieval from the DB will work. RA decides that original goal of “retrieving data” can no longer be achieved, and it reports the exception to EHA with the context that “partial failure leading to data not retrievable and on-going conversation ID = 0104”. EHA upon receiving the report, evaluates the exception with its context, and realises that this RA is not fit to answer queries. The EHA then instructs DA to modify the service metadata repository to change the status of RA from functional to non-functional.

6.4.3 Semantic Model for Exception-Fault Relations and Process Description

Now that the overall structure and workflow of the exception handling model has been clarified, the semantic models for the service process descriptions and the exception-fault relations can be explained. The many-to-many exception-fault relations have been modelled in an ontology that includes the concept of an exception context. Figure 40 shows part of the ontology and how the key concepts interlink with each other. For instance every exception has one or more corresponding handlers, each of which is linked to specific actions. Actions are then embedded in a relevant service process.

The manner in which the specific exception handling actions are embedded into service process descriptions is now examined. In defining the services an agent supports, it is necessary to not only identify the service interfaces, but also clarify the service compositions. As it is mentioned previously that service is defined using

OWL-S, and the composition is modelled as a composite process. Here a composite process can be considered as equivalent to the composite tasks in HTN.

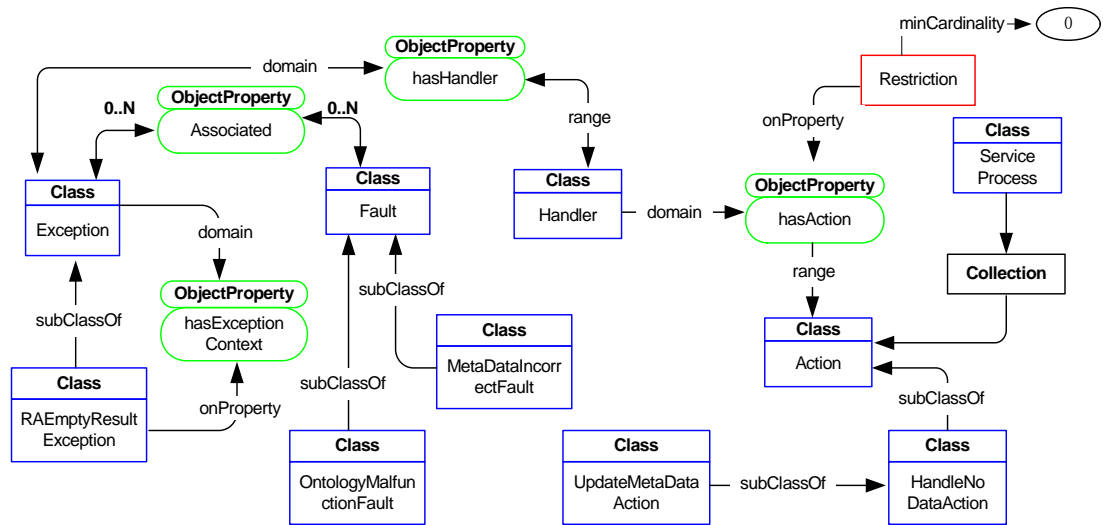


Figure 40 Exception-Fault Ontology

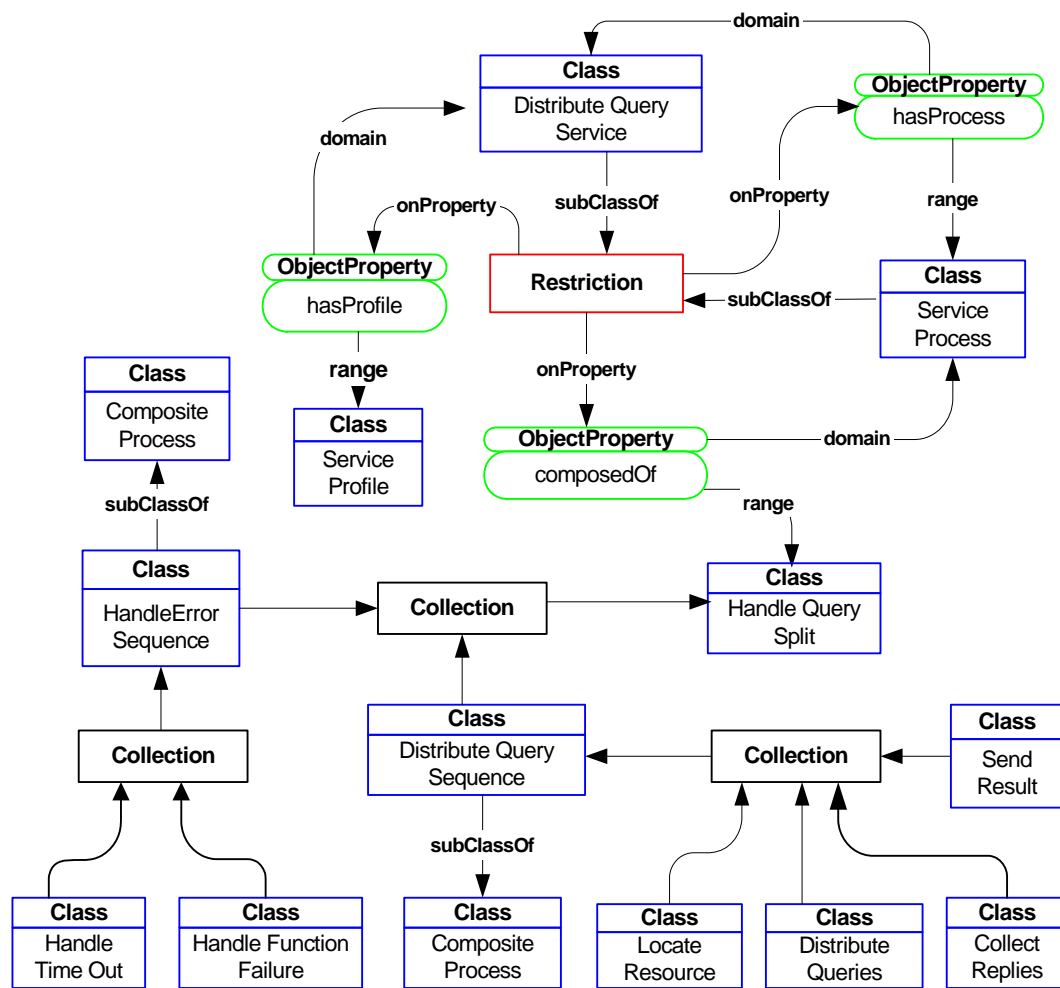


Figure 41 Service description and process model in OWL-S

An example of one of Task Agent's services "Distribute Query Service" has been decomposed into two separate sequences, namely the distribute query sequence and handle error sequence. The ontological bindings of the service, process, and sequence concepts in this instance are shown in Figure 41. The figure illustrates the modelling of level 1 task decomposition. Level 2 decomposition requires input from the plan library, which defines that "query local metadata" and "query directory" are equivalent to achieving goal "locate resource". In other words, these two plans can be used interchangeably, and it is up to the planning unit to choose which to use.

6.4.4 Heuristic Knowledge Modelling

Heuristics are accumulated through experience, and are referred to as rules of thumbs. In handling exceptions, human experts or software developers usually through the experience of interacting, such as debugging and testing with the system accumulate a collection of rules of thumbs, which then serve as guidelines and instructions to direct the handling process. The rules of thumbs together with human brain's ability to reason under imprecision, uncertainty and complications, enable human experts to deploy more accurate judgement about the exceptional occurrence and to make more effective decisions. These rules of thumbs represent heuristic knowledge and are usually modelled as rules.

Some generic illustrative EH rules are given as follows:

1. If a failure interrupts the original plan to the goal, then evaluate if there is an alternative to the goal.
2. If an alternative plan exists, then evaluate if it also fits any constraints in the query.
3. If the alternative plan does not fit the query constraint, then change the goal to one that fits the requirement.

Rules are implemented using JESS, the Java Expert System Shell [63]. This can be easily integrated into Java programs. An example of a rule defined in JESS is shown in Figure 42, and the rule specifies that if a RA partial failure occurs that leads to incomplete data then it changes the current goal into "reply incomplete data".

```

(defrule change-current-goal
(currentGoal RetrieveCompleteData)
(exception (name RAPartialFailure) (type ?exceptionType) (effect ?exceptionEffect)
(OBJECT ?exceptionObject))
(constraint (value ?constraintValue))
(IncompleteData ?exceptionEffect)
=>
(bind ?currentGoal ReplyIncompleteData))

```

Figure 42 A rule in JESS that shows the change of the current goal

6.4.5 Metadata Assisted Recovery Mechanism

6.4.5.1 Recovery from Total Agent Failure

In case an agent fails completely, JADE [57] provides a method called “take down” for the agent to carry out any necessary outstanding tasks before it terminates. Using the “take down” method, an agent can log the metadata of active interaction and exception context, as well as informing the EHA of the exception context, e.g., the file name where the metadata for the exception context is stored. The file name is later passed onto the substitute agent that retrieves the metadata and resumes the interrupted interaction. This approach is more effective than conventional redundancy, because of the use of interaction metadata for the recovery. It is also more efficient than replication, because both active and passive replications require the replica to be operational in parallel which produces a considerable overhead for the system.

To decide whether or not to recover the failed agent’s functions using a substitute or to carry on regardless, the EHA collects the exception context and decides. The exception context mainly contains the following information: the estimated effect of the exception, the last active conversation(s) of the failed agent, and the constraint of the current query processing session. Suppose one of the resource agents fails during a query session, and there is a very stringent time constraint to answer the user query, the EHA decides to reply with incomplete results, rendering the system function

degradation, rather than revoking a substitute which would take a longer time to reply with the complete results. When the query constraints allow, the EHA prioritises the handling method that provides complete results. This means the substitute agent is used to resume the interrupted conversation and to achieve the original goal.

The interaction metadata helps the substitute agent to restore itself to the last state before the agent fails. AIPs are modelled as transition tables, and each possible state are included in the model, so that the interaction metadata only needs to store the last state of the conversation. The substitute agent can directly carry on from the stored state with the conversation, without needing to repeat previous message exchanges.

6.4.5.2 Recovery of Partial Failures

When only part of the agent function is faulty, it is not necessary to use substitute every time, especially when other functions are working normally. For the case of a partial agent failure, the agent's monitor unit detects the failure and evaluates its effect and reports to its planning unit and EHA. The planning unit evaluates whether the current goal is still achievable in the presence of the failure. If the goal is achievable with alternative paths, then set the alternative as current path. If the current goal is no longer achievable, then set a new goal. Upon receiving the failure report, the EHA evaluates the overall situation of the query processing session. It first of all examines the active agent conversations at the time of the exception event, and then the effect of the exception on the overall query processing.

For example if the link between a resource agent and the data source breaks, the monitor unit of the resource agent detects the exception and assesses it, and the effect of the exception is non-deterministic. The worst case is that the break-down is permanent and no more data can be retrieved from the data source. Now the planning unit in the resource agent re-evaluates the situation and decides whether or not the original goal of "retrieve complete set of data" is achievable in the face of the exception and can set a new goal "reply with empty result". At the same time the EHA evaluates the overall situation and because the non-deterministic effect of the exception and the failure renders the resource agent not capable of answering any

queries, it can decide to inform Directory Agent to change the status of the resource agent into inactive.

6.5 Evaluation

To evaluate the Exception Handling framework, several test cases are identified to demonstrate the operation of the key components. The Directory Agent is one of the key agents in the system, and the first test case is to show how a total directory agent failure affects the system and how the system copes with this kind of failures. The second test case shows the recovery mechanism from a total TA failure using substitutes. Finally partial Resource Agent failures that cause the system functions to degrade are handled. Each of these scenarios is described in detail in the following sections.

6.5.1 Total Directory Agent Failure (Recoverable Failure)

There are two sub-scenarios here with a total directory agent failure, one is that directory agent is not available when other agents start up, and the other is that directory agent suddenly fails during conversation.

Here the total DA failure is categorised as a recoverable failure because the DA functions can be achieved with alternative plans, either using the local metadata repository or broadcasting to locate the service in the system. These two alternatives have been modelled in the plan library, and when directory agent is not available for metadata query, the agent switches to one of these alternatives depending on the specific circumstance.

Consider the following scenario that a Task Agent starts up, and one of the goals it has is to publish its service descriptions with a Directory. This goal needs to be translated into an executable plan. There are two plan templates defined in the plan library, which are to “find Directory Agent, and send a register request”, providing the directory is available, and “wait for Directory announcement, and send register request” if the directory is not available. The Task Agent needs to identify if the Directory Agent is available to select the plan. If the Task Agent queries the platform DF, and the search result for Directory Agent is empty, then it chooses a second plan

template. In this way, the unavailability of the DA at start-up does not prevent the Task Agent from performing other tasks. In other words, the availability of the DA is not a prerequisite for the system to function. Now that a Resource Agent also starts up, and finds out that the Directory Agent is not available and postpones the service register request. However it detects there is another agent in the system, and it initiates communication using a Hello request. Task Agent replies with information of itself, and establishes the contact with the Resource Agent. The peer interaction using the Hello Protocol thus enables the population of local metadata repositories without the mediation of the Directory. The local metadata structure further strengthens the robustness of the metadata query mechanism.

If the Directory Agent now comes online, and realises there are two agents (Task Agent and Resource Agent) in the system, it broadcasts an announcement. Agents in receiving the announcement, reply with requests to register their service descriptions. The communication to exchange metadata is thus established.

Consider what happens if the Directory Agent suddenly goes offline during a conversation, the DF detects the disappearance and informs the EHA. The EHA then informs other agents engaged in the conversation about the failure. Agents then trigger their re-planning unit to select an alternative plan for the metadata query, either to query a local repository or to make a broadcast, depending on their conditions.

This scenario illustrates the flexibility of planning, re-planning and its direct consequence on the enhancement of system robustness: both Directory unavailability at start-up and total directory failure during conversation can be fully recovered using alternative plans.

6.5.2 Total Task Agent Failure

Consider the following scenario: UA1 initiates a conversation with ID=001 that is a UC1 query to TA3. The subsequent interaction includes TA3 forwarding the query to RANERI and RAIOW which has been identified to have the relevant data, then RANERI and RAIOW respectively replies to TA3 with the data set. However there

is no record of a reply being sent to UA1 by TA3 with conversation ID=001 in the metadata repository in MA. UA1 reports a time-out exception, which triggers the exception handling process.

In this case upon receiving the exception reporting the total failure of TA3, EHA initiates the recovery mechanism immediately. EHA also broadcasts the failure message to agents that have previously been in contact with TA3 so that they are informed of the situation. In case, any of them that are waiting for a reply, they can choose whether to stop waiting and carry out other tasks, or to wait till the substitute agent resumes the interrupted interaction.

The total failure recovery mechanism using substitutes proposed here makes use of the agent conversation metadata to resume conversation that the “failed” agent was last engaged in, more efficiently. Since the conversational context is recovered from the metadata, it is possible to resume the interrupted conversation without starting all over again. The mechanism greatly improves the efficiency of failure recovery, in reducing any information exchange needed to restore agents into the states just before being interrupted. This is especially useful when two parties have engaged in a conversation spanning over a long time. Consider the following scenario where a task agent is requested to manage multiple query sessions for the multi-dimensional view agent. The task agent collects all the results from the first batch of sub-queries and distributes the second batch sub-queries and waits for the final results, then a total failure occurs causing it to stop functioning completely. Without an audit record of the metadata for all these previous interactions, a substitute agent has to repeat all of them at start up, before it can answer the requester.

The interaction metadata of the failed agent is only shared with the substitute. The mechanism works as follows: interaction metadata for the currently active conversation is stored in a permanent storage, i.e. a file. In the (JADE) take-down method the agent not only deregisters itself with DF, but also passes on the file name to EHA; and EHA then informs the selected substitute of the file name so that the stored metadata can be used. With the metadata sharing mechanism, a substitute can immediately be restored to the state before the exception and be ready to process any reply messages coming in.

Now that the death of TA3 has been established, EHA immediately broadcasts the request for a substitute TA to finish the query processing, using the TA list from DA. The first TA that replies to the broadcast message is appointed as the substitute, e.g., TA1. Figure 43 shows the agent interactions to identify substitute agent.

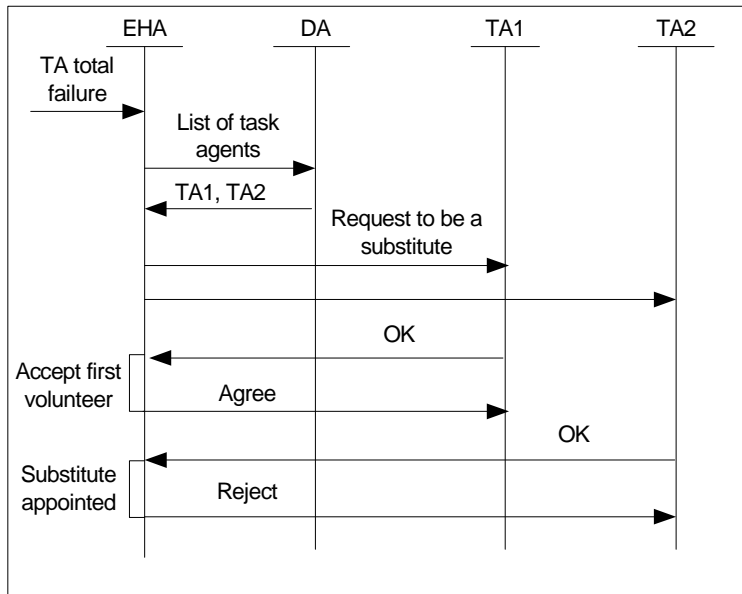


Figure 43 Interaction to identify a substitute

Now that TA1 agrees to finish the query processing left by TA3, the EHA informs RANERI and RAIOW that the exception handling has started and requests their cooperation. For simplicity we assume all agents are benevolent and cooperative, therefore RANERI and RAIOW would always agree to cooperate with the EH. Now the MA sends out the instruction for RANERI and RAIOW to resend their replies for conversation 001 to TA1 with message ID of RANERI012U003 and RAIOW002U005 respectively. RANERI and RAIOW check their local message cache, retrieve the messages and send them again. TA1 receives the replies and starts the post-processing. TA1 finishes the processing by informing UA1 the final data set. The metadata on both agent interaction and message content are described in the previous chapter. Figure 44 shows the conversation being resumed.

This scenario shows a typical use of substitutes in handling total agent failures. In some cases invoking a substitute to resume the lost functions can completely recover

the failure. However in other cases, system functions have been degraded because substitute invocation is not efficient enough.

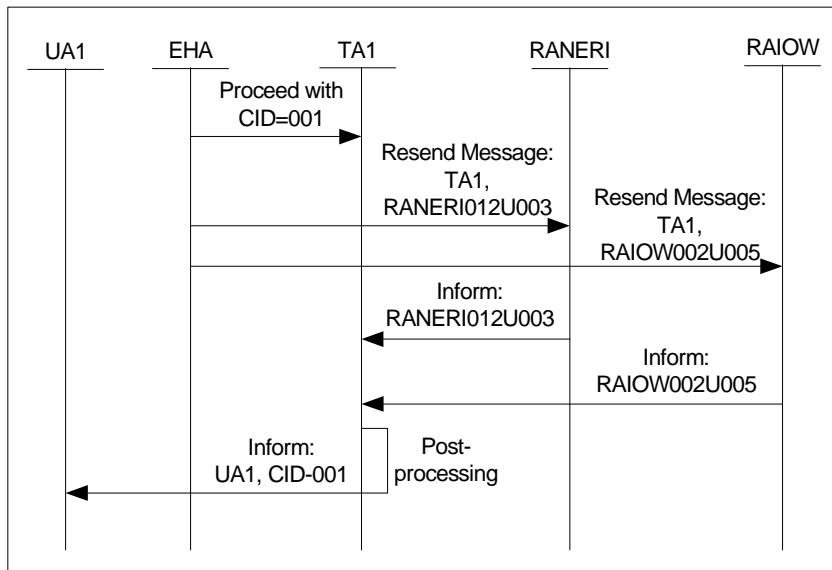


Figure 44 Interaction to resume query processing conversation

6.5.3 Partial Resource Agent Failure (degradable failure)

A query is distributed to two data sources and one of the resource agents fails, which directly leads to the incompleteness of the results. There are two handling methods: if the query has a very stringent time constraint, then it replies with partial results. If the time constraint is not a pressing issue, a substitute is invoked to resume the processing. Both of the recovery mechanisms, with and without substitutes, have been described in previous sections 6.4.5.1 and 6.4.5.2, and are not repeated here. The reasoning processes in these recovery mechanisms are much more complex than those in the previous scenarios and involve the exception context, system level semantics and query constraints. This scenario mainly highlights how the decisions are made with the reasoning.

6.5.4 Description of Experiments

Two experiments have been conducted to test the system. The first experiment is to test how the Multi-Agent System reacts to the unavailability of DA at start-up and its

total failure during a conversation. In other words, this experiment tests the robustness of the system against total agent failures, together with the flexibility of the distributed directory structure. The second experiment is to test the reaction to partial agent failures, in particular partial failures of Resource Agents.

The main ideas of both experiments are to simulate the exceptions in the above scenarios and to observe how the system can cope with them and whether the query processing functions are interrupted when exceptions occur or not. However providing evidence of a partially functioning system can be more challenging than showing a complete failure. In an information retrieval system, the most direct result of a system functioning normally is that data is retrieved to answer queries (which is similar to the results shown in chapter five). From the end users' point of view, the exceptions would not have existed during the query process. At a more detailed level, various outputs can be used to show that agents cooperate in case of exception and that the system core functions are maintained. For instance the JADE sniffer agent monitors agent interactions of the system, and sequences of message exchange can be observed at the monitor interface, which is a valuable insight of the operation of MAS.

6.5.4.1 Experiment one: Directory Agent failure

There are two parts of the experiment: the first part shows the sequence of a task agent, a resource agent and the directory agent that starts up one by one in order; the second part shows that agents use alternative plans to circumvent complete Directory Agent failure during a conversation.

The first part of the experiment shows that agents can start up without the presence of the Directory Agent and that they establish contact with each other using the Hello Protocol. When the Directory Agent becomes available, it broadcasts an announcement so that agents can send their request to register service descriptions. It is clear at this point that dependency on the Directory Agent at start-up has been eliminated. Some outputs such as the agent's reasoning processes and the message exchange monitored at the sniffer agent have been enclosed in the Appendix (see section 10.1).

Now the Directory Agent starts up, it searches the system DF and finds out that there are two agents available, i.e. TA and RA. The DA then sends an announcement to them. Upon receiving the announcements, TA and RA reply to the DA with a request to register their service descriptions. The DA thus builds up the metadata repository of agents' service descriptions.

The second part of the experiment shows the sudden disappearance of the DA, which is simulated as an agent death. In this case a dummy agent is used to simulate the user agent. It sends a query to TA, which triggers the metadata query. The TA first of all prioritises the plan of querying directory for metadata. When it checks with DF, it finds out that DA is not available. The TA now goes into the re-planning process, it notices the original plan of query DA for metadata is not achievable because the DA is down, and it checks the plan library, and finds out there is an alternative, i.e. to broadcast for the service. Since broadcasting is identified as the current plan, the TA then collects all the available agents' addresses from DF and sends out the broadcast request for a specific type of resource agent. The resource agent that matches the requirement then replies to the broadcast message, and the query processing continues. The second part of the experiment shows that agents circumvent total DA failure by using alternative plans to obtain metadata and resumes query processing without a centralised directory.

This experiment demonstrates that the system is robust in handling a total Directory agent failure with the use of a decentralised metadata structure and Hello Protocol.

6.5.4.2 Experiment two: resource agent failure

Consider the case when the link between resource agent and DB breaks. This leads to a situation that the RA is not able to retrieve any data out of a DB. The DB is one of the data sources being queried in the system, and this exception leads to system function degradation. This experiment shows how degradable fault tolerance is maintained, when one of the RA to DB links breaks down during a multi-query session. When TA sends the data query out, RA detects there is an exception in the link to DB. RA immediately assesses the situation and the exception handling follows. The RA realises the break-down has serious consequences that no more data

can be retrieved while the exception persists. In addition, it cannot tell whether the break-down is transient or permanent. In the worse case scenario, it is permanent and this needs to be addressed. After reasoning about the context and potential effects, the RA reports the exception and its consequence to the EHA. The RA also informs other agents that have been in communication with it about this exception. EHA assesses the exception situation, because the worst case is that this RA is not capable of answering any data queries, EHA decides to inform the Directory Agent to change the RA's status into inactive (to avoid sending any more queries to RA). This handling step is usually not achieved if only local exception handling is applied, because it requires the reasoning at the system level that oversees the consequences and constraints between processes. Then RA replies with an empty result to TA, so that the exception would not cause non-deterministic delay. Upon receiving the empty result and together with the previous exception notification, the TA reasons that best action is to reply with partial results (results from other resource agents) to the end user. The output of the agent reasoning processes and message exchange sequences at the sniffer agent is available in Appendix (see section 10.2).

The second experiment shows a partial resource agent failure scenario where reasoning at the system level is needed. It demonstrates the system robustness against partial agent failures, and more importantly the reasoning at the system level prevents failure propagation.

6.6 Discussion

The architectural overview in Figure 34 shows the addition of the new Exception Handling Agent with an EH ontology. This is the main focus of this Chapter. The EHA is not a stand-alone agent. It interacts with other agents to collect exception reports with context, and gives instructions on exception handling strategies. Also the monitoring function of individual agent has been extended to detect exceptions and interacts with the planning unit.

The fundamental feature of fault tolerance is the system operation in the presence of exceptions and failures. Therefore it is crucial to distinguish the effects, on the system any exceptions or failures can have before enacting the handling. Here effects

are categorised into recoverable, degradable and hard failures. The categorisation simplifies the handling process, so that the appropriate strategies are identified accordingly.

The proposed EH framework can be seen as a combination of centralised and distributed approaches, because it has a central EH agent as well as individual agents' monitoring and re-planning functions. Planning however is not always an optimal solution to complex problems, because it may take too much time and changes can happen quickly, especially, in a dynamic open environment. One extreme case is that the decision unit may spend all the time re-planning and no action is actually taken. Rules in the case are essential so that not every single exception occurrence requires a complete re-planning. This also promotes the efficiency of exception handling.

The EH framework has been demonstrated to improve the service availability, by reducing the single point of failure, adding alternative paths to achieving a goal, and by supporting recovery mechanisms for both total and partial agent failures. Therefore the requirement of fault-tolerance for query management is considered to be fulfilled.

6.7 Summary

Fault-tolerance in particular query related exceptions have been tackled in this chapter. An exception handling framework is proposed to handle both total agent failures and partial agent failures. Agents are equipped with the monitoring and re-planning functions to help them detect and handle exceptions efficiently. An Exception Handling Agent acts as the central reasoning unit to control the exception handling workflow, and it collects exception context system-wide, subsequently making decisions about what it regards as the right handling strategy. There are two parts to the knowledge base for the exception handling, one is the semantic descriptions of service processes with embedded specific exception handling workflows, and the other is the exception-fault relations. The reasoning process combines both parts in the knowledge base as well as exception context to identify the handling measure. Total agent failures are handled using substitutes with

interaction metadata that assist substitutes to resume communication from the point of interruption. Partial failures are masked and alternative plans are adopted to achieve the original goal or a sub-optimal goal, if the original one is no longer achievable. Experiments were carried out to validate the framework, in particular two cases have been tested, one for a recoverable failure and one for a degradable failure.

7 Conclusion and Further Work

7.1 Contributions

The specific problem domain of managing heterogeneous Inland Water (IW) quality information to support multiple user viewpoints and the general problem and challenges of developing distributed information systems have been analysed, leading to a specific set of problem properties and requirements for distributed information systems for the IW domain. A critical survey of the state of the art has been undertaken to assess the strengths and limitations of existing systems to meet these problem challenges.

Based upon the problem properties and requirements, a hybrid multi-agent system framework has been developed and evaluated. This supports coordinated distributed user data queries on virtual data resources which are in turn executed as multiple related data queries on the actual individual heterogeneous database resources. The framework adopts an Ontology approach to heterogeneous data interoperability and to allow different virtual views of the same data. The agent model itself principally offers support for coordinating the distributed query processes. The multi-agent system framework also supports a virtual data warehouse model to enable multi-dimensional derived user views of the data resources.

The multi-agent system framework incorporates a semantic metadata repository or directory service to aid the virtualisation and selection of information resources. The directory service is designed to be decentralised so that it can be accessed locally by each agent and this improves its fault-tolerance. It supports a user-centred view of service in terms of quality of service as well as a provider centred view of services. Metadata is gathered and maintained to improve coordination of distributed queries to improve the fault-tolerance of interactions.

The multi-agent system includes a semantic fault-tolerance system that supports both substitution and fault recovery. It uses semantic propagation of low-level systems errors so that in some cases it can be more meaningfully handled at the application level.

Although, some of the research stems out from the EU FP5 IST EDEN-IW Project, the work shown in the thesis is entirely the author's own contribution. The distinction between the author's input and other project partners' contribution into the EDEN-IW project has been clearly stated in chapter four, section 4.6.1.

7.2 Conclusion

This thesis has described the challenges and complexities of answering information queries within an open distributed information domain that consists of changing heterogeneous data resources, evolving applications processes and multi-level and multi-dimensional data views.

An experimental framework to solve these challenges has been researched, developed and validated. The chosen architecture focuses on the support of data interoperability utilising semantics in the form of Ontologies and mappings to align semantically, heterogeneous data models to a common virtual semantic data model. The distribution and coordination of data queries across autonomous resources is realised using a multi-agent system. In addition, virtual multi-dimensional data views and fault-tolerance are also supported in the framework. Such a hybrid design of MAS has not previously been specified or applied to facilitate distributed heterogeneous database resource access and management.

In order to validate the operation of the framework, it has been applied to the problem of determining the quality of inland water in a pan-European environment. The framework is developed and evaluated in phases, which has been detailed in chapters four, five and six.

As it is a complex solution to a set of complex problem requirements, it is worth discussing the reusability and applicability of the solution to other problem domains. First of all, the solution consists of several components such as a domain semantic model, metadata model, and multi-agent system. One of the main advantages of a computational semantic model using standard languages is the reusability across applications. However it requires input from domain experts to capture the key relations between domain concepts. Also the development of a domain semantic model is an iterative process, requiring refinement and modification. The details of

the semantic model development cycle have been covered in [130]. However once the key structure of the domain ontologies has been identified, it is reasonably extensible in that new terms can be easily incorporated into the model without a complete re-engineering of the model.

Secondly, the role-based methodology for agent design is generic, meaning the general agent structure can be applied to other problem domains. The key issue involved here is the integration of agents and the semantic model, which means that agents may need some specific reasoning processes for the domain-specific conceptualisation. Therefore the solution with the hybrid Multi-Agent and semantic architecture can be considered as re-usable in other domains, providing that input from domain experts is available for the creation of the semantic model. This is one of the major benefits of the solution framework.

A second main benefit is the flexibility it provides, to tolerate heterogeneities and to promote data interoperability. The advantages of a virtual data warehouse compared to the physical approach have been stressed previously (see section 2.2.2). Incorporating semantic descriptions of the MDDB structure into a virtual approach adds another layer of flexibility, as changing the semantic relations of the MDDB structure does not have any actual effect on the data storage. Also different MDDB structures can be applied to the same physical data storage if needed without conflicting with each other.

However there are limitations. Obviously it is a complex framework to design and develop, therefore the cost is considerable. The iterative nature of the semantic model development is one of the major costs incurred. Interfacing the metadata model and agents, i.e. fine tuning the agents to process the semantics in the exact way expected, can be difficult and time-consuming. In addition, the inherent costs of developing and testing distributed system may not be an optimal choice for some problems that do not require a high level of query processing flexibility to incorporate heterogeneous data sources.

A domain property of the Inland Water data is that data values once generated do not change very frequently, and in most cases for query processing, the raw data values

do not change between the time the query is issued and results are retrieved. Subsequently the framework has not considered the real-time data value update as a requirement for query processing. For some applications such as stock markets and other financial applications, the query processing model needs to be modified to handle real-time data updates.

7.3 Further Work

There are several extensions to the MAS framework that can enhance the solution as described in the following sections.

7.3.1 Agent Social Acquaintance Model

Currently, the agent's local directory, only stores the contact details of previous service providers they interacted with. They do not make use of the metadata of the individual service invocation sessions. One natural extension is to build up a more comprehensive model of the previously used service providers and contacts and to share with others of the similar interests, to support a community of practice model. In particular, agents could rate their service sessions by more subjective criteria such as how helpful the service is in providing information. This service rating is complementary to the service quality metadata and helps users to choose the right service provider using additional usability criteria. Subsequently, profiles of the service providers can be built, using both quality measurement metadata and user rating metadata. These profiles are a key part of a richer social acquaintance model.

An agent social acquaintance model was first proposed in the ARCHON project to capture the meta-level information of cooperative agents [61]. Other researchers [13] [75] have furthered identified the cooperative metadata with task structures. The agent acquaintance model can be extended to include the use of service invocation metadata and user rating as well. Firstly, specific parameters for service rating are identified: responsiveness, helpfulness, and reasonability. One deciding factor to determine the helpfulness of a service provider depends on whether or not it gives detailed reasons when it needs to refuse a request and possible suggestions for future interaction. Secondly these service usage profiles are stored and managed by the

individual agents in a local repository that they control the access to. Thirdly, users with similar interests form groups where these profiles are shared and are used to assist service selection. Here, a similarity of interests is defined based on mainly their query preferences, for example, users that request data from a French DB with stringent time constraints. Groups are formed so that agents know that metadata shared by group members is more relevant to their own needs than non-group members' metadata. Leaders are cooperatively given the responsibility for approving applications to join the group, to act responsibly as part of the group and to leave it.

7.3.2 Orchestrate Services to Service Level Agreement

The distributed query orchestration presented in the framework has taken into account the quality of service so that the selected service orchestration is optimised according to the utility function of the query requester. The utility presented mainly concerns the completeness of the result and efficiency of the query processing, which are respectively translated into the capabilities and quality of service in service providers' terms. This homogeneous utility model needs to be extended to accommodate different user requirements, and the Service Level Agreement (SLA) is introduced for this purpose. SLA is a formal negotiated agreement between two parties, that is to say, it is a contract that exists between customers and their service providers. The current service quality semantic model can be extended to include semantic definitions of SLA related concepts and properties. A SLA item can be defined as a combination of required properties. Another aspect of having a mutual agreement between users and service providers is that penalties could be applied if the providers fail to deliver the agreed level of service. A direct consequence of such penalties is that the credibility of the service provider decreases.

7.3.3 Enhance Fault-tolerance with Probabilistic Risk Analysis

The current semantic model for the fault-tolerance framework defines the relations between exceptions (effects) and faults (causes) but their many-to-many relationship is to a degree ad hoc. This model can be extended to include probability information about the likelihood of each exception occurring. Using quantifiable information

about the exception and its context, the exception handling could be made more precise and efficient.

A systematic assessment of the exceptions involved is needed in the first instance. Probabilistic Risk Assessment or PRA [131] is a systematic and comprehensive methodology to evaluate risks associated with a complex engineered technological entity. Risk in PRA is defined as a feasible detrimental outcome of an activity or action. Two main quantities characterise risks: the magnitude (severity) of the possible adverse consequences, and the likelihood (probability) of occurrence of each consequence. Generally PRA is used to find out: the potential components of causing undesirable consequences, the potential detrimental effects, and the probability of the undesirable consequences occurring. Two common methods of answering these questions are Event Tree Analysis and Fault Tree Analysis. In a fault tree, the root or top event of a tree of logic is an undesirable effect. For each situation that can cause the effect, a series of logic expressions is added to the tree. When fault trees are labelled with actual numbers of probabilities, failure probabilities can be calculated from the trees.

Currently the magnitude of risks has been captured with the exception-fault ontology, which associates exceptions with their potential effects. Their likelihood however has not been defined. Therefore fault trees need to be specified and instantiated with the list of exceptions in the system. Then probabilities of each exception can be calculated. The reasoning process takes into account both the magnitude (reasoning result from the exception ontology) and the likelihood (calculation result from the fault trees) to decide a suitable handling strategy.

8 Author's Publications

Conferences and journal publications

1. Stjernholm, M., Poslad, S., Zuo, L., Sortkjær, O. and **Huang, X.** 2004, *The EDEN-IW Ontology Model for Sharing Knowledge and Water Quality Data between Heterogeneous Databases*, EnviroInfo2004.
2. Poslad, S., Tan, J.J., **Huang, X.** and Zuo, L. 2005, *Middleware for semantic-based security and safety management of open services*, Int. J. Web and Grid Services, Vol. 1, Nos. 3/4, pp. 305-327.
3. Zuo, L., Poslad, S. and **Huang, X.** 2006, *Agent-based Information Sharing and Semantic Information Retrieval from Heterogeneous Databases*, International Conference on Business Knowledge Management, Macao, China.

Book Chapters

1. Poslad, S., Stjernholm, M., Zuo, L. and **Huang, X.** 2007, *Review of Models and Technologies for Database Integration*. In: Environmental Data Exchange Network for Inland Water, Haastrup P. and Wurtz J. (Eds.), Elsevier, ISBN 978-0-444-52973-2, pp. 51-67.
2. Poslad, S., Zuo, L. and **Huang, X.** 2007, *Multi-agent System Technology in Distributed Database Systems*. In: Environmental Data Exchange Network for Inland Water, Haastrup P. and Wurtz J. (Eds.), Elsevier, ISBN 978-0-444-52973-2, pp. 97-121.
3. Stjernholm, M., Poslad, S., Zuo, L., Sortkjar, O. and **Huang, X.** 2007, *An Ontology-based Approach for Enhancing Inland Water Information Retrieval from Heterogeneous Databases*. In: Environmental Data Exchange Network for Inland Water, Haastrup P. and Wurtz J. (Eds.), Elsevier, ISBN 978-0-444-52973-2, pp. 123-143.

9 Bibliography

- [1] Aberer, K., Datta, A. and Hauswirth, M. 2004, *Efficient, Self-Contained Handling of Identity in Peer-to-Peer Systems*, in IEEE Transactions on Knowledge and Data Engineering, Vol. 16, No. 7, pp. 858-869.
- [2] Agent Unified Modelling Language or AUML, website: <http://www.auml.org/>, last accessed on 2007-04.
- [3] Allcock, B., Chervenak, A., Foster, I., Kesselman, C. and Livny, M. 2005, *Data Grid tools: enabling science on big distributed data*, in Journal of Physics: Conference Series, 16, pp. 571-575.
- [4] Antoniou, G. and Harmelen. F. 2004, *A Semantic Web Primer*, MIT Press. ISBN 978-0-262-01210-2.
- [5] Avizienis, A., Laprie, J., Randell, B. and Landwehr, C. 2004, *Basic concepts and taxonomy of dependable and secure computing*, IEEE Transactions on Dependable and Secure Computing, Vol. 1, No. 1, pp. 11–33.
- [6] Baeza-Yates, R. and Ribero-Neto, B. 1999, *Modern Information Retrieval*, Addison-Wesley ISBN 0-201-39829-X.
- [7] Berners-Lee, T., Hendler, J. and Lassila, O. 2001, *The Semantic Web*, Scientific American, Vol. 284, No. 55, pp. 28-37.
- [8] Botelho, L., Willmott, S., Zhang, T. And Dale, J. 2002, *Review of Content Languages Suitable for Agent- Agent Communication*, Agentcities.RTD project deliverable, retrieved from webpage: ic2.epfl.ch/publications/documents/IC_TECH_REPORT_200233.pdf, last accessed on 2007-04.
- [9] Breiter, P. and Sadek, M. D. 1996, *A Rational Agent as a Kernel of a Cooperative Dialogue System: Implementing a Logical Theory of Interaction*. Proceedings of ECAI-96 Workshop Agent Theories, Architectures, and Languages, pp. 261-276.
- [10] Bronstein, A. et. al, 2001, *Self-aware service: using Bayesian networks for detecting anomalies in Internet-based services*, in Proceedings of 2001 IEEE/IFIP International Symposium on Integrated Network Management, pp. 623-638.
- [11] Brugali, D. and Sycara, K. 1998, *Agent technology: a new frontier for the development of application networks?* In M. Fayad, D. Schmidt, and R.

Johnson, editors, *Object-Oriented Application Frameworks*. New York: John Wiley and Sons.

- [12] Burns, A. and Wellings, A., 1990, *Real-Time Systems and Their Programming Languages*, Addison-Wesley, 1990, ISBN 0-201-17529-0.
- [13] Cao, W., Bian, C-G., and Hartvigsen, G. 1997, *Achieving Efficient Cooperation in a Multi-Agent System: The Twin-Base Modelling*. in Proceedings of Co-operative Information Agents, Kandazia, P. and Klusch, M. eds., pp. 210-221, LNAI No. 1202, Springer Verlag, Heidelberg.
- [14] Caromel, D. and Chazarain, G. 2005, *Robust Exception Handling in an Asynchronous Environment*, in ECOOP Workshop on Exception Handling in Object Oriented Systems: Developing Systems that Handle Exceptions, number 05-050 in Technical Reports – Laboratoire d’Informatique, de Robotique et Micro-Electronique de Montpellier.
- [15] Castro, M. and Liskov, B. 1999, *Practical Byzantine Fault Tolerance*, Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI), pp. 173-186.
- [16] Chen, D. and Cohen, R. 2002, *AERO: An Outsourced Approach to Exception Handling in Multi-agent Systems*, In Proceedings of the 15th Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence, pp. 16-30.
- [17] Clark, K. 2004, RDF Data Access Use Cases and Requirements, W3C. Retrieved from webpage: <http://www.w3.org/TR/rdf-dawg-uc/>, last accessed on 2007-04.
- [18] Corcho, O., Fernandez-Lopez, M. and Gomez-Perez, A. 2003, *Methodologies, tools and languages for building ontologies: where is their meeting point?* In Data & Knowledge Engineering, Vol. 46, Issue 1, pp. 41-46.
- [19] Damasceno, K., Cacho, N., Garcia, A., Romanovsky, A. and Lucena, C. 2006, *Context-Aware Exception Handling in Mobile Agent Systems, The MoCA Case*, in Proceedings of the 2006 International Workshop on Software Engineering for Large-scale Multi-Agent Systems, pp. 37-44.
- [20] Decker, K. S. and Lesser, V. R. 1992, *Generalizing the partial global planning algorithm*. International Journal of Intelligent and Cooperative Information Systems, 1(2), pp. 319-346.

- [21] Decker, K., Lesser, V., Nagendra Prasad, M. V. and Wagner, T. 1995, *MACRON: An Architecture for Multi-agent Cooperative Information Gathering*, in Proceedings of the CIKM'95 Workshop on Intelligent Information Agents.
- [22] Decker, K. and Sycara, K. 1997, *Intelligent Adaptive Information Agents*, Journal of Intelligent Information Systems, 9, 239-260.
- [23] Decker, S., Melnik, S., Harmelen, F. V., Fensel, D., Klein, M., Broekstra, J., Erdmann, M. and Horrocks, I. 2000, *The Semantic Web: The Roles of XML and RDF*, IEEE Internet Computing October 2000, Vol. 15, nr. 3 pp. 63-74.
- [24] Dellarocas, C., Klein, M., and Rodriguez-Aguilar, J. A. 2000, *An Exception-Handling Architecture for Open Electronic Marketplaces of Contract Net Software Agents*, in Proceedings of the 2nd ACM conference on Electronic commerce, pp. 225-232.
- [25] Denker, G., Kagal, L., Finin, T., Paolucci, M. and Sycara, K. 2003, *Security for DAML web services: annotation and matchmaking*, The SemanticWeb - ISWC-2003, Lecture Notes in Computer Science, Vol. 2870/2003, pp. 335-350.
- [26] Desmedt, Y. 1994, *Threshold cryptography*, European Transactions on Telecommunications, Vol. 5, No. 4, pp. 449-457.
- [27] Dong, X. et al, 2003, *Autonomia: an autonomic computing environment*, in Proceedings of the 2003 IEEE International Performance, Computing, and Communications Conference, pp. 61-68.
- [28] EDEN-IW project, IST-2000-29317, project home page: <http://www.eden-iw.org>, last accessed on 2007-04.
- [29] Erol, K., Hendler, J., and Nau, D. S. 1994, *Semantics for Hierarchical Task Network Planning*, Technical Report CS-TR-3239, UMIACS-TR-94-31, Computer Science Dept., University of Maryland.
- [30] Everest, G. 1986, *Database Management: objectives, system functions, and administration*. New York, NY: McGraw-Hill. ISBN 978-0070197817.
- [31] Extensible Mark-up Language XML, Home page: <http://www.w3.org/XML/>, last accessed on 2007-04.
- [32] Fan, Y. and Gauch, S. 1999, *Adaptive Agents for Information Gathering from Multiple, Distributed Information Sources*, in Proceedings of 1999 AAAI Symposium on Intelligent Agents in Cyberspace, pp. 40-46.

- [33] Fedoruk, A. and Deters, R. 2002, *Improving Fault-Tolerance by Replicating Agents*, AAMAS'02, pp. 737-744.
- [34] Fensel, D. 2004, *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*, Springer, ISBN 978-3540416029.
- [35] Ferber. J. 1999, *Multi-Agent Systems*. Addison-Wesley. ISBN: 978-0201360486.
- [36] Filho, A. H., Staa, A., and Lucena, C. 2003, *A Component-Based Model for Building Reliable Multi-Agent Systems*, In Proceedings of the 28th Annual NASA Goddard Software Engineering Workshop (SEW'03), pp. 41-50.
- [37] Finin, T., Labrou, Y. and Mayfield, J. 1995, *KQML as an agent communication language*, MIT Press, Cambridge. Retrieved from webpage: <http://www.flacp.fujitsulabs.com/~yannis/publications/mitpress96.pdf>, last accessed on 2007-04.
- [38] FIPA, Foundation for Intelligent Physical Agents, Home page: <http://www.fipa.org>, last accessed on 2007-04.
- [39] FIPA Agent Management Specification, 2004, retrieved from webpage: <http://www.fipa.org/specs/fipa00023/SC00023K.html>, last accessed on 2007-04.
- [40] FIPA ACL Message Structure Specification, 2002, retrieved from webpage: <http://www.fipa.org/specs/fipa00061/SC00061G.html>, last accessed on 2007-04.
- [41] FIPA RDF content language specification, 2001, retrieved from webpage: <http://www.fipa.org/specs/fipa00011/XC00011B.html>, last accessed on 2007-04.
- [42] Fowler, J., Perry, B., Nodine, M. and Bargmeyer, B. 1999, *Agent-Based Semantic Interoperability in InfoSleuth*, SIGMOD Record 28:1, pp. 60-67.
- [43] Friedman-Hill E. 2003, *Jess in Action: Java Rule-Based Systems (In Action series)*. Manning Publications, ISBN 1930110898.
- [44] Garay, J.A., Gennaro, R., Jutla, C. and Rabin, T. 2000, *Secure distributed storage and retrieval*, Journal of Theoretical Computer Science, Vol. 243, No. 1-2, pp. 363-389.
- [45] Genesereth, M. R., Keller, A. M. And Duschka, O. 1997, *Infomaster: An Information Integration System*, in Proceedings of 1997 ACM SIGMOD Conference, pp. 539-542.

- [46] Giampapa, J. A. and Sycara, K. 2002, *Team-Oriented Agent Coordination in the RETSINA Multi-Agent System*, technical report CMU-RI-TR-02-34, Robotics Institute, Carnegie Mellon University.
- [47] Goodenough, J. B. 1975, *Exception Handling: Issues and a Proposed Notation*, in *Communications of the ACM*, Vol. 18, Issue 12, pp. 683-696.
- [48] Greenwood, D. 2005, *JADE Web Services Integration Gateway*, retrieved from webpage: http://x-opennet.org/netdemo/Demos2005/aamas2005_netdemo_2pg.pdf, last accessed on 2007-04.
- [49] Haastrup, P. and Wurtz, J. (eds) 2007, *Environmental Data Exchange Network for Inland Water*, Elsevier, ISBN: 978-0-444-52973-2.
- [50] Haegg, S. 1996, *A Sentinel Approach to Fault Handling in Multi-Agent Systems*, in *Proceedings of the Second Australian Workshop on Distributed AI*, in conjunction with the Fourth Pacific Rim International Conference on Artificial Intelligence (PRICAI'96), pp.181-95.
- [51] Halpin, T. 2001, *Information Modelling and Relational Databases, From Conceptual Analysis to Logical Design*, Morgan Kaufmann publisher, ISBN 1-55860-672-6.
- [52] Harmelen, F. and Fensel, D. 1999, *Practical Knowledge Representation for the Web*, in *Proceedings of the IJCAI'99 Workshop on Intelligent Information Integration*.
- [53] Haverkamp, D. S. and Gauch, S. 1998, *Intelligent Information Agents: Review and Challenges for Distributed Information Sources*, *Journal of the American Society of Information Science*, Vol. 49, No. 4, pp. 304-311.
- [54] Hoile, C., Wang, F., Bonsma, E. and Marrow, P. 2002, *Core Specification and Experiments in DIET: A Decentralised Ecosystem-inspired Mobile Agent System*, in *Proceedings of AAMAS'02*, pp. 623-630.
- [55] Huhns, M. N. and Singh, M. P. 2005, *Service-Oriented Computing: Key Concepts and Principles*, *IEEE Internet Computing*, Vol. 9, Issue 1, pp. 75-81.
- [56] InfoSleuth Agent System, Project Home page: <http://www.argreenhouse.com/InfoSleuth/>, last accessed on 2007-04
- [57] JADE, Java Agent DEvelopment Framework, Home page: <http://sharon.cselt.it/projects/jade/>, last accessed on 2007-04.

- [58] Jalote, P. 1994, *Fault Tolerance in Distributed Systems*, P T P Prentice Hall, Englewood Cliffs, New Jersey, ISBN 0-13-301367-7.
- [59] Jena, A Semantic Web Framework for Java, home page: <http://jena.sourceforge.net/>, last accessed on 2007-04.
- [60] Jennings, N. R. 1995, *Controlling Cooperative Problem Solving in Industrial Multi-Agent Systems using Joint Intentions*, Artificial Intelligence. 75(2), pp. 195-240.
- [61] Jennings, N. R., Corera, J. and Laresgoiti, I. 1995, *Developing Industrial Multi-Agent Systems (Invited Paper)*. In Proceedings of 1st International Conference on Multi-Agent Systems (ICMAS '95), pp.423-430, San Francisco, USA.
- [62] Jennings, N. R., Sycara, K., Wooldridge, M. 1998, *A road map of agent research and development*, Autonomous Agents and Multi-Agent Systems, 1 (1), pp. 7-38.
- [63] JESS, Java Expert System Shell, home page: <http://herzberg.ca.sandia.gov/jess/>, last accessed on 2007-04.
- [64] KAON, the Karlsruhe Ontology and Semantic Web Tool Suite (KAON), home page: <http://kaon.semanticweb.org/>, last accessed on 2007-04.
- [65] KIF, Knowledge Interchnage Format, home page: <http://logic.stanford.edu/kif/kif.html>, last accessed on 2007-04.
- [66] Klein, M. and Dellarocas, C. 1999, *Exception Handling in Agent Systems*, Autonomous Agents 1999, pp. 62-68. Seattle, WA, USA.
- [67] Klein, M. and Dellarocas, C. 2000, *A Knowledge-based Approach to Handling Exceptions in Workflow Systems*, In Journal of Computer Supported Cooperative Work (CSCW), Vol. 9 No. 3-4, pp. 399-412.
- [68] Klein, M. Rodriguez-aguilar, J-A. and Dellarocas, C. 2003, *Using Domain-Independent Exception Handling Services to Enable Robust Open Multi-Agent Systems: The Case of Agent Death*, Autonomous Agent and Multi Agent Systems, 7, pp. 179-189.
- [69] Klusch, M. 2001, *Information agent technology for the internet: a survey*, Data & Knowledge Engineering 36 (3) pp. 337-372.
- [70] Kumar, S., and Cohen, P. R. 2000, *Towards a Fault-Tolerant Multi-Agent System Architecture*, International Conference on Autonomous Agents,

- Proceedings of the fourth international conference on Autonomous agents, pp. 459-466.
- [71] Lamport, L., Shostak, R. and Pease, M. 1982, *The Byzantine generals problem*, ACM Transactions on Programming Languages and Systems, Vol. 4, No. 3, pp. 382–401.
- [72] Langley, B. K., Paolucci, M. and Sycara, K. 2003, *Discovery of Infrastructure in Multi-Agent Systems*, in Proceedings of AAMAS'03, pp. 1046-1047.
- [73] Lesser, V., Horling, B., Raja, A., Zhang, X. and Wagner, T. 2000, *Resource-Bounded Searches in an Information Marketplace*, IEEE Internet Computing, March April 2000, pp. 49-58.
- [74] Malkhi, D. and Reiter, M. 2000, *An architecture for survivable coordination in large distributed systems*, IEEE Transactions on Knowledge and Data Engineering, Vol. 12, No. 2, pp. 187–202.
- [75] Marik, V., Stepankova, O. and Pechoucek, M. 2001, *Acquaintance model for integration-oriented multi-agent systems*, in Proceedings of the International Conference of the Chilean Computer Science Society SCCC 2001, pp. 186-192.
- [76] Milanovic N., Malek M. 2004, *Current Solutions for Web Service Composition*, IEEE Internet Computing, Vol. 8, Issue 6, pp. 51-59.
- [77] Nicol, D., Sanders, W. and Trivedi, K. 2004, *Model-based evaluation: from dependability to security*, IEEE Transactions on Dependable and Secure Computing, Vol. 1, No. 1, pp. 48–65.
- [78] Nodine, M., Perry, B. and Unruh, A. 1998, *Experience with the InfoSleuth Agent Architecture*, In Proceedings of AAAI-98 Workshop on Software Tools for Developing Agents.
- [79] Nodine, M., Bohrer, W. and Ngu, A. H. H. 1999, *Semantic Brokering over Dynamic Heterogeneous Data Sources in InfoSleuth*, In Proceedings of the International Conference on Data Engineering, pp. 358-365.
- [80] Nodine, M., Fowler, J., Ksiezzyk, T., Perry, B., Taylor, M. and Unruh, A. 2000, *Active Information Gathering In InfoSleuth*, International Journal of Cooperative Information Systems, Vol. 9, No. 1-2, pp. 3-28.

- [81] Odubiyi, J.B. et al. 1997, *SAIRE-A Scalable Agent-Based Information Retrieval Engine*, in Proceedings of the first International conference on autonomous agents, pp. 292-299.
- [82] O'Hare, G. M. P. and Jennings, N. R. 1996, *Foundations of Distributed Artificial Intelligence*, Wiley. ISBN: 978-0-471-00675-6.
- [83] OKBC, Open Knowledge Base Connectivity, home page: <http://www.ai.sri.com/~okbc/>, last accessed on 2007-04.
- [84] OntoWeb, Ontology-based information exchange for knowledge management and electronic commerce, home page: <http://www.ontoweb.org/>, last accessed on 2007-04.
- [85] OWL Web Ontology Language, home page: <http://www.w3.org/2001/sw/WebOnt>, last accessed on 2007-04.
- [86] Padgham, L. and Lambrix, P. 2000, *Agent Capabilities: Extending BDI Theory*, In Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, pp. 68-73.
- [87] Perry, B., Taylor, M. and Unruh, A. 1999, *Information Aggregation and Agent Interaction Patterns in InfoSleuth*, in Proceedings of Conference on Cooperative Information Systems, pp. 314-324.
- [88] Peterson, J. L. *Petri Net Theory and the Modelling of Systems*, Prentice-Hall. ISBN: 978-0136619833.
- [89] Platon, E. Honiden, S. and Sabouret, N. 2006, *Challenges in Exception Handling in Multi-Agent Systems*, in Proceedings of International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, SELMAS'06, pp. 45-50.
- [90] Poslad, S., Charlton P. 2001, *Standardizing agent interoperability: the FIPA approach*. In: Michael Luck, Vladimír Marík, Olga Stepánková, Robert Trappl (Eds.): Multi-Agent Systems and Applications, 9th ECCAI Advanced Course, ACAI 2001 and Agent Link's 3rd European Agent Systems Summer School, EASSS 2001, Prague, Czech Republic, Selected Tutorial Papers. Lecture Notes in Computer Science 2086 Springer 2001, ISBN 3-540-42312-5, pp. 98-117.
- [91] Poslad, S., Stjernholm, M., Zuo, L., Huang, X. *Review of Models and Technologies for Database Integration*. In: Environmental Data Exchange

- Network for Inland Water, Hastrup P., Wurtz J. (Eds), Elsevier, ISBN 978-0-444-52973-2, 2007, pp. 51-67.
- [92] Poslad, S., Zuo, L. and Huang, X. 2007, *Multi-agent System Technology in Distributed Database Systems*. In: Environmental Data Exchange Network for Inland Water, Hastrup P. and Wurtz J. (Eds.), Elsevier, ISBN 978-0-444-52973-2, pp. 97-121.
- [93] Powers, S. 2003, *Practical RDF*, O'Reilly. ISBN: 0-596-00263-7.
- [94] Preece, A. 1999, *COVERAGE: verifying multiple-agent knowledge-based systems*, in Knowledge-Based Systems 12 (1997) pp. 37-44.
- [95] Russell, S. and Norvig, P. 2002, *Artificial Intelligence A Modern Approach, second edition*, Prentice Hall, ISBN: 0-13-080302-2.
- [96] Resource Description Framework RDF, Home page: <http://www.w3.org/RDF/>, last accessed on 2007-04.
- [97] Seaborne, A. 2004, *RDQL-A Query Language for RDF*, Online only, retrieved from: <http://www.w3.org/Submission/RDQL/>, last accessed on 2007-04.
- [98] Searle, J.R. 1969, *Speech Acts*. Cambridge University Press, 1969. ISBN: 0521071844.
- [99] Shah, N. H., Chao, K-M., Anane, R. and Godwin, N. 2003, *A Flexible Approach to Exception Handling in Open Multi-Agent Systems*, In Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-03) Challenges'03 Workshop, pp. 7-10, 2003.
- [100] Shah, N., Chao, K-M., Godwin, N., Younas, M. and Liang, C. 2004, *Exception Diagnosis in Agent-Based Grid Computing*, IEEE Conference on Systems, Man and Cybernetics, pp. 3213-3219.
- [101] Shanmugasundaram, J. et al. 1999, *Relational Databases for Querying XML Documents: Limitations and Opportunities*, in Proceedings of the 25th VLDB conference, pp. 302-314.
- [102] Sheth, A., Ramakrishnan, C. and Thomas, C. 2005, *Semantics for the Semantic Web: The Implicit, the Formal and the Powerful*, Int'l Journal on Semantic Web & Information Systems, 1(1), pp. 1-18.
- [103] Singh, M. P. and Huhns, M. N. 2005, *Service-Oriented Computing: Semantics, Processing, Agents*, John Wiley & Sons, ISBN: 0-470-09148-7.

- [104] Souchon, F., Dony, C., Urtado, C. and Vauttier, S. 2003, *A Proposition for Exception Handling in Multi-Agent Systems*, In: Proceedings of the 2nd international workshop on Software Engineering for Large-Scale Multi-Agent Systems, pp. 136-143.
- [105] Spyns, P., Meersman, R., Jarrar, M. 2002, *Data modelling versus Ontology Engineering*, in Proceedings of Database and Information Systems Research for Semantic Web and Enterprises Workshop, special section on Semantic Web and Data Management, pp. 12-17.
- [106] Steinacker, A., Ghavam, A. and Steinmetz, R. 2001, *Metadata Standards for Web-Based Resources*, IEEE Multimedia, Vol. 8, Issue 1, pp. 70-76.
- [107] Sterritt, R. and Bustard, D. 2003, *Autonomic Computing – a means of achieving dependability?* in Proceedings of the 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, pp. 247-251.
- [108] Stjernholm, M., Poslad, S., Zuo, L., Sortkjar, O. and Huang, X. 2007, *An Ontology-based Approach for Enhancing Inland Water Information Retrieval from Heterogeneous Databases*. In: Environmental Data Exchange Network for Inland Water, Haastrup P. and Wurtz J. (Eds.), Elsevier, ISBN 978-0-444-52973-2, pp. 123-143.
- [109] SWAD-Europe, Semantic Web Advanced Development for Europe, project home page: <http://www.w3.org/2001/sw/Europe/>, last accessed on 2007-04.
- [110] Sycara, K. and Zeng, D. 1996, *Coordination of Multiple Intelligent Software Agents*, International Journal of Cooperative Information Systems 5(2/3) pp.181-211.
- [111] Sycara, K. and Zeng, D. 1996, *Multi-Agent Integration of Information Gathering and Decision Support*, In proceedings of ECAI96 12th European Conference on Artificial Intelligence. pp. 549-556.
- [112] Sycara, K., Paolucci, M., Van Velsen, M. and Giampapa, J. 2003, *The RETSINA MAS Infrastructure*, Autonomous Agents and Multi-Agent Systems, 7, pp. 29-48.

- [113] Tan, J.J., Poslad, S. 2004, *Dynamic Security Reconfiguration in Semantic Open Services Environment*. Journal on Engineering Applications of Artificial Intelligence, vol. 17, no. 7, pp. 783-797.
- [114] Tan, J.J., Poslad, S., Titkov, L. 2006, *A Semantic Approach to Harmonising Security Models for Open Services*. Journal of Applied Artificial Intelligence, Vol. 20, No. 2-4, pp. 353-379.
- [115] Thain, D. and Livny, M. 2002, *Error Scope on a Computational Grid: Theory and Practice*, Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing HPDC-11, pp. 199-208.
- [116] Tohma, Y. 2002, *Fault Tolerance in Autonomic Computing Environment*, in Proceedings of the 2002 Pacific Rim International Symposium on Dependable Computing (PRDC'02), pp. 3-6.
- [117] Toivonen S. and Niemi, T. 2004, *Describing Data Sources Semantically for Facilitating Efficient Creation of OLAP Cubes*, in Proceedings of the 3rd International Semantic Web Conference ISWC2004.
- [118] Universal Description, Discovery, and Integration, home page: <http://www.uddi.org/>, last accessed on 2007-04.
- [119] Varakantham, P. R., Ganwani, S. K., and Karlapalem, K. 2001, *On Handling Component and Transaction Failure in Multi Agent Systems*, ACM SIGecom Exchanges, Vol.3 Issue 1, pp. 32-43.
- [120] Vdovjak, R. and Houben, G-J. 2001, *RDF-Based Architecture for Semantic Integration of Heterogeneous Information Sources*, in Proceedings of Workshop on Information Integration on the Web, pp. 51-57.
- [121] Watson, R. 2002, *Data Management Databases and Organisations*, Third Edition John Wiley & Sons, Inc. ISBN: 0-471-41845-5.
- [122] Web Intelligence consortium, home page: <http://wi-consortium.org/>, last accessed on 2007-04.
- [123] Weyns, D., Vizzari, G. and Holvoet, T. 2006, *Environment for Situated Multi-agent Systems: Beyond Infrastructure*, E4MAS 2005, LNAI 3830, pp. 1-17, 2006.
- [124] Weyns, D., Parunak, H. V. D., Michel, F., Holvoet, T. and Ferber, J. 2005, *Environments for Multiagent Systems, State-of-the-Art and Research Challenges*, in Proceedings of the First International Workshop on Environments for Multiagent Systems, LNAI 3374, pp. 1-47.

- [125] Wooldridge, M., Jennings, N. R. and Kinny, D. 2000, *The Gaia Methodology for Agent-Oriented Analysis and Design*, Autonomous Agents and Multi-Agent Systems, Vol. 3, Issue 3.
- [126] Wooldridge, M. 2001, *An Introduction to MultiAgent Systems*, Wiley, ISBN: 0 471 49691 X.
- [127] W3C, home page: <http://www.w3.org>, last accessed on 2007-04.
- [128] Xu, P. and Deters, R. 2004, *MAS & Fault-Management*, Proceedings of the 2004 International Symposium on Applications and the Internet (SAINT'04), pp. 283-286.
- [129] Zuo, L. and Poslad, S. 2003, *Supporting multi-lateral semantic information viewpoints when accessing heterogeneous distributed environmental information*. 1st European workshop on multi-agent systems, EUMAS.
- [130] Zuo, L. 2006, *A Semantic and Agent-Based Approach to Support Information Retrieval, Interoperability and Multi-Lateral Viewpoints for Heterogeneous Environmental Databases*, PhD Thesis. Retrieved from: <http://www.elec.qmul.ac.uk/networks/documents/ZUO-Landong-PhDthesis.pdf>, last accessed on 2007-04.
- [131] Probabilistic Risk Assessment, website: http://en.wikipedia.org/wiki/Probabilistic_risk_assessment, lass accessed on 2007-04.

10 Appendix: Experiment Output for the Fault-tolerance Framework

10.1 Experiment one: DA failure

When Directory Agent is not available at start up, other agents use Hello Protocol to initialise communication. Figure 45 shows the message exchange monitored at the sniffer agent between Task Agent and Resource Agent using Hello Protocol.

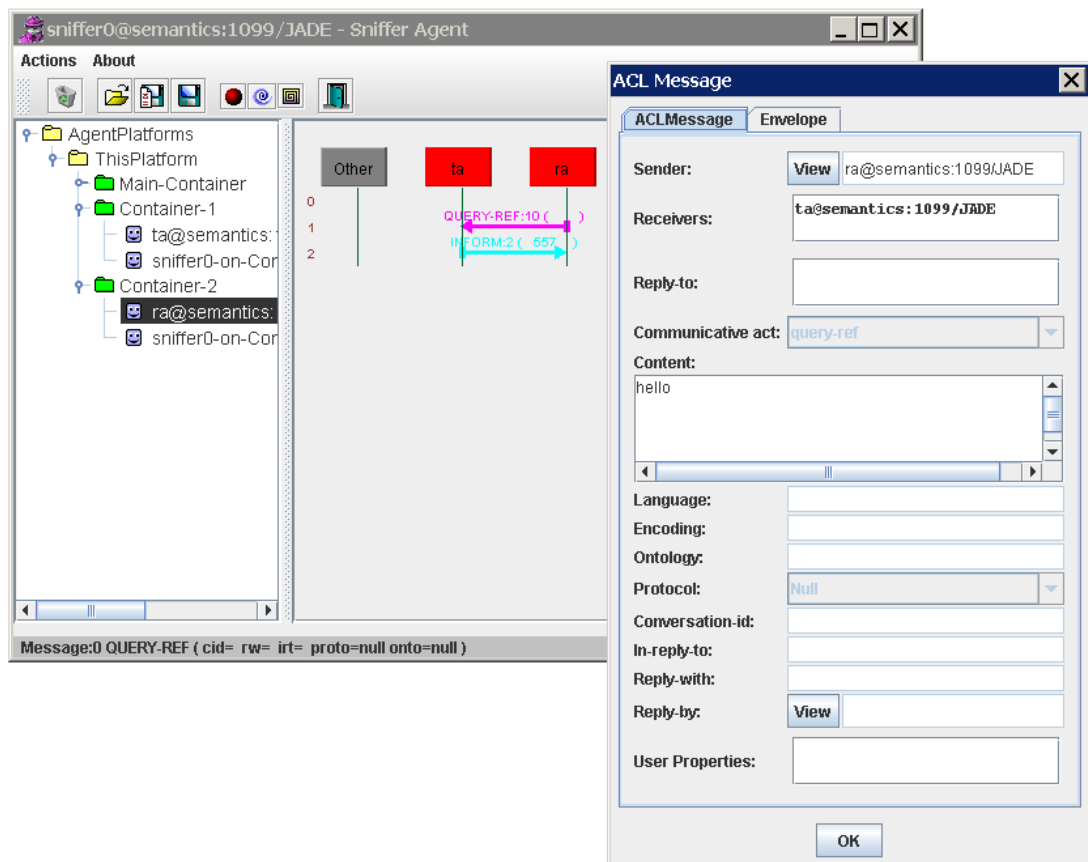


Figure 45 Hello message exchange between TA and RA

Figure 46 and Figure 47 show the reasoning process for TA and RA, respectively, in this scenario, that TA decides to wait for incoming messages and RA sends a Hello request to TA.

```

Select C:\WINNT\system32\cmd.exe
-----
22-Mar-2007 16:10:40 jade.core.BaseService init
INFO: Service jade.core.management.AgentManagement initialized
22-Mar-2007 16:10:40 jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
22-Mar-2007 16:10:40 jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
22-Mar-2007 16:10:40 jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
22-Mar-2007 16:10:41 jade.core.AgentContainerImpl joinPlatform
INFO: -----
Agent container Container-1@JADE-IMTP://semantics is ready.
-----
at finally in init function
default DF name: df
default AMS name: ams
agentName = ta
finally executed.
agent alive
no other agent available at the moment, do nothing.
simple behaviour added, ready to receive messages.
-

```

Figure 46 TA reasoning process

```

Select C:\WINNT\system32\cmd.exe
INFO: Service jade.core.management.AgentManagement initialized
22-Mar-2007 16:10:53 jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
22-Mar-2007 16:10:53 jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
22-Mar-2007 16:10:53 jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
22-Mar-2007 16:10:54 jade.core.AgentContainerImpl joinPlatform
INFO: -----
Agent container Container-2@JADE-IMTP://semantics is ready.
-----
at finally in init function
default DF name: df
default AMS name: ams
agentName = ta
**an element added to available agent list**
finally executed.
agent alive
**availableAgentList is not null
---available agent list does not contain da---
preparing to send hello message to other agents available...
hello message set.
hello message sent.
simple behaviour added, ready to receive messages.

```

Figure 47 RA reasoning process

Now the Directory Agent comes online, and broadcasts its existence. Figure 48 shows the announcement messages being sent. Subsequently both TA and RA send requests to register their service descriptions.

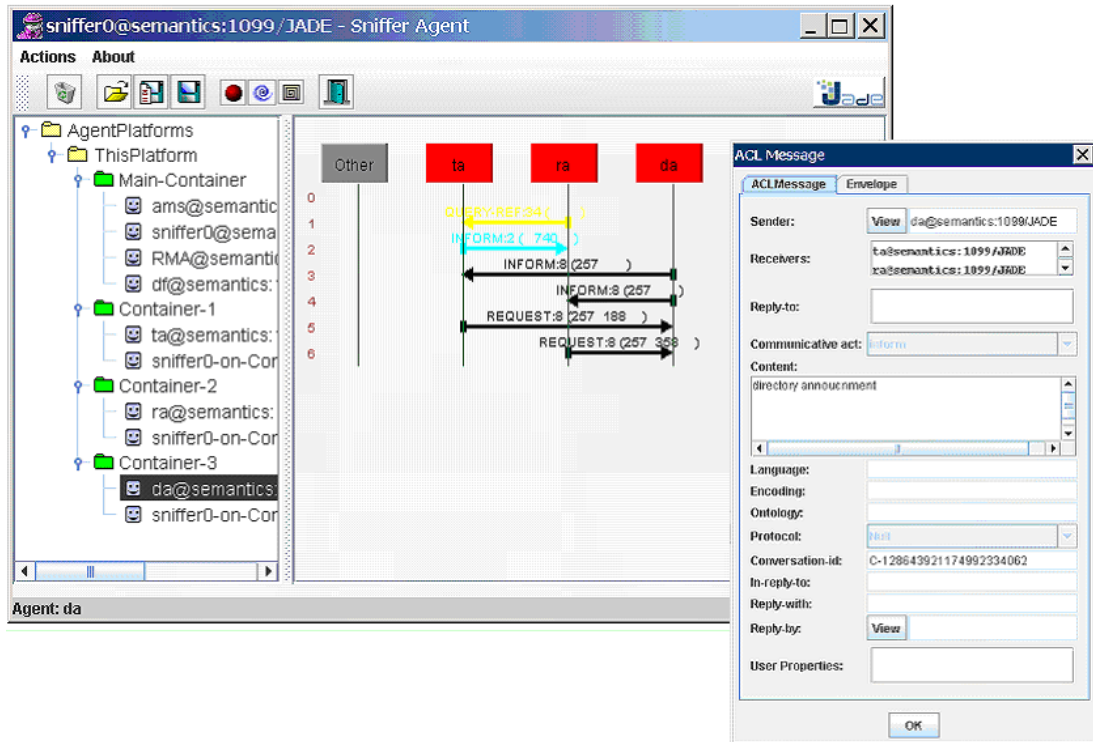


Figure 48 DA broadcasts announcement

TA needs to find the relevant data sources to handle the incoming query, but DA is not available. The reasoning process of the TA is shown in Figure 49.

```

C:\ Select C:\WINNT\system32\cmd.exe
finally executed.
agent alive
no other agent available at the moment, do nothing.
simple behaviour added, ready to receive messages.
inMsgSenderName==ra0
inMsgSenderName==ra1
inMsgSenderName==da
---received a msg from da!---
inMsgSenderName==da0
checked DF, and directory agent is not available
now need to broad-cast for the resource agent
broad-cast message sent
inMsgSenderName==ra0

```

Figure 49 TA reasons about what to do when directory is not available

The last three message exchanges given in Figure 50 shows that the resource agent that matches the request replies to the broadcast message. Subsequent interactions continue between them to finish the conversation.

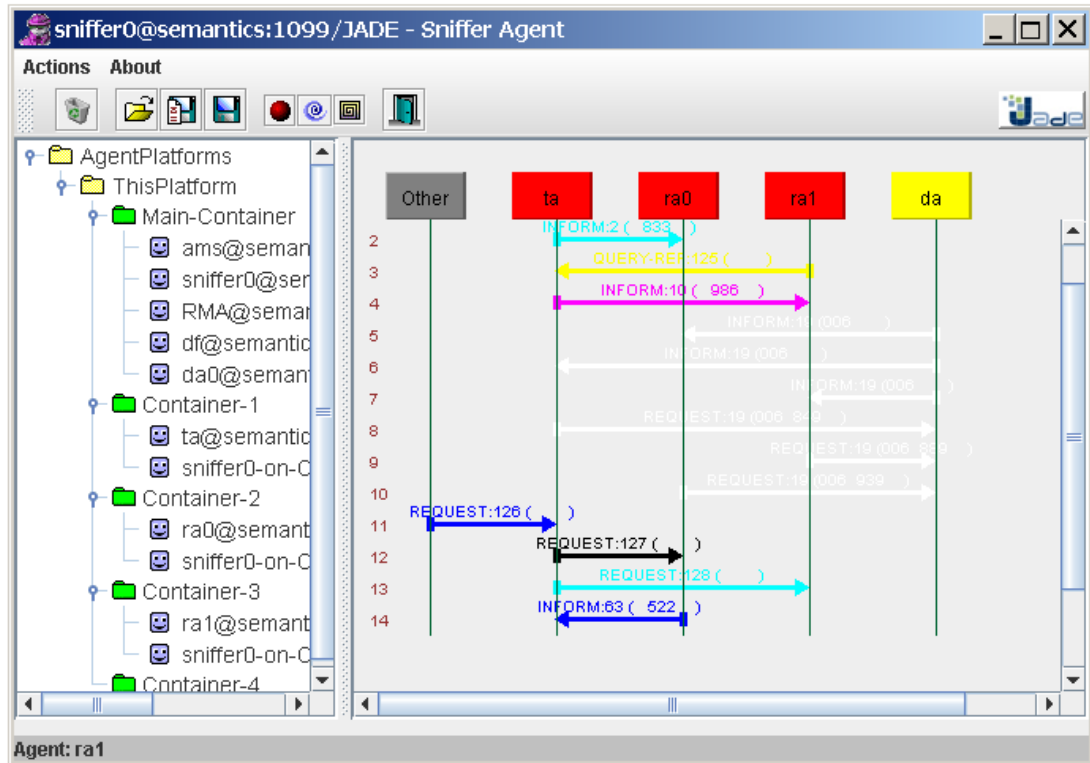


Figure 50 TA broadcasts for a specific resource agent

10.2 Experiment two: RA failure

Resource Agent detects a partial failure that interrupts the processing of all data queries. Figure 51 and Figure 52 show the reasoning process in RA and EHA respectively in case of the exception event. RA decides to inform EHA about the failure and its consequence, and inform the participants of the conversation as well. EHA decides it is necessary to inform the Directory Agent so that no more data queries are sent to the RA while this exception persists.

```
Select C:\WINNT\system32\cmd.exe
13-Apr-2007 18:44:53 jade.core.AgentContainerImpl joinPlatform
INFO: -----
Agent container Container-3@JADE-IMTP://semantics is ready.
-----
directory is online, preparing to send a register message
register message sent
received the reply from directory about service registration
registration OK
receive a data query from TA
---exception detected: communication link to the DB is broken---
report to EHA immediately
report sent
reply to the current conversation about this failure
reply sent
```

Figure 51 Exception handling steps in RA

```
Select C:\WINNT\system32\cmd.exe
13-Apr-2007 18:45:03 jade.core.AgentContainerImpl joinPlatform
INFO: -----
Agent container Container-4@JADE-IMTP://semantics is ready.
-----
directory is online, preparing to send a register message
register message sent
register successful
received an exception report from RA
the recovery of the exception is non-deterministic, and RA is not functional
need to inform directy to update RA's status
preparing the message for directory
inform to directory sent
```

Figure 52 Reasoning in EHA upon receiving RA's exception report

The overall agent interaction is shown in Figure 53.

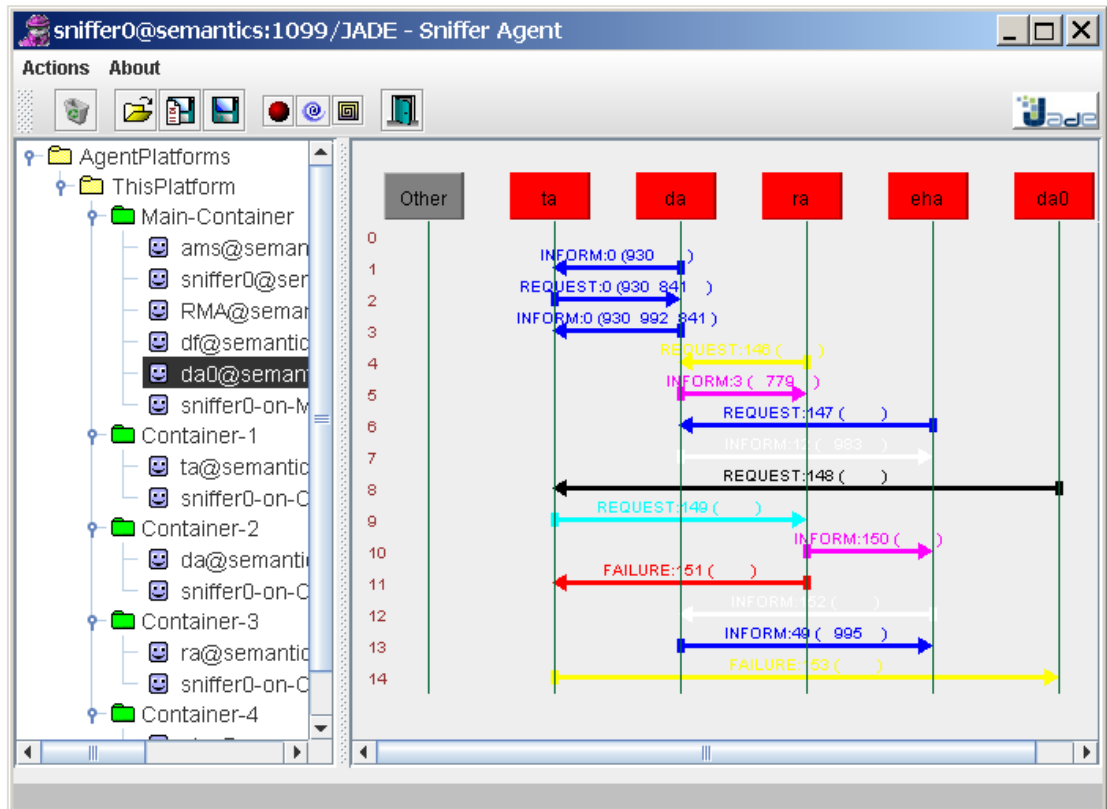


Figure 53 Agent interaction in handling RA exception