

# **An Architecture for Searching Radio Signals**

*coordination of task and result sharing in the search process*

**Mark J. Ronkes**

Master's Thesis

Department of Computer Science

Open Universiteit Nederland

April 2008



### **Graduation committee**

prof. dr. Lex Bijlsma	chairman	(Open Universiteit Nederland)
prof. dr. Johan T. Jeuring	supervisor	(Universiteit Utrecht, Open Universiteit Nederland)
dr. Bastiaan J. Heeren	supervisor	(Open Universiteit Nederland)

#### *Advisory members*

ir. Sylvia Stuurman	expert	(Open Universiteit Nederland)
---------------------	--------	-------------------------------

### **Student info**

Student name	Mark J. Ronkes
Student number	837168392

## Acknowledgements

First of all, I would like to thank prof.dr. Johan Jeuring and dr. Bastiaan Heeren for excellent supervision and useful feedback. In addition, I would like to thank ir. Sylvia Stuurman for valuable advice, feedback and reading comments.

Second, very special thanks have to be said to Barry van de Lustgraaf for the fruitful discussions and support. In addition, I would like to thank Henk Kloosterman for advice, help and guidance during my research project. Moreover, I am grateful to Marco van Maanen, a fellow student, but above all a good friend, for the fruitful discussions.

A final word of thanks to my family and friends for their understanding, patience and support, each in their own unique way. Last, but certainly not least, I would like to thank Dagmar for her understanding, patience, and never ending support during all these years of study.

# Summary

## *Background*

Searching for radio signals in the ether with the currently used approaches and resources, has turned out to be inefficient. Due to the complexity of the search process, execution of similar or overlapping search tasks often occurs. Practical experience has shown that a lot of signal systems are not fully deployed. Nowadays, due to the lack of integration possibilities, knowledge acquired on the basis of the search process is only partially captured and manual tasks are to relatively large extent needed for signal searching. Ultimately, this has resulted in the current situation where the search process is performed mainly manually. In order to improve the efficiency, we aim at deploying autonomous processes.

## *Methods*

To define the selection criteria for coordination mechanisms and model selection, we have used architectural drivers. For design of the architecture we used tactics and architectural patterns. Documentation of the architecture is based on the key stakeholders/concerns and evaluation of this architecture has been performed with scenarios and measuring techniques. The JADE (Java Agent DEvelopment framework) and TuCSoN (Tuple Centre Spread Over the Network) middleware have been used to develop prototypes.

## *Results*

We defined maintainability, reliability, and performance as the architectural drivers. Workflow-, negotiation-, and multi agent planning mechanisms were selected to coordinate task sharing activities. A reactive coordination model was selected to coordinate the communication between entities that share search related information with different structures. In order to meet the key quality attributes, the layers style has been selected for search task execution and the blackboard style for sharing search related information. Two prototypes have been developed for a Proof of Concept: SearchOne, providing search task services to users in the search process and ShareOne, providing exchange of markers between entities that are decoupled in space and time, in a publish/subscribe manner. The architecture is described with model-, process-, shared data-, and deployment views.

---

### *Conclusions*

The designed architecture enables both autonomous and efficient execution of search tasks and sharing of markers between several systems and users, resulting in improved efficiency of the search process. Maintainability and reliability were considered key concerns to enable an automatic and efficient search process, whereas performance was considered to be a less essential concern. The designed architecture mainly addresses the key concerns.

The developed (extensive) prototypes provide a lot of functionality and suitable interfaces with the most important signal systems, but lack some minor functionalities to directly deploy these prototypes in practice. However, essential parts of the prototypes can be directly deployed to obtain the desired improvement of efficiency of the search process.

# Table of contents

<b>Summary</b>	<b>i</b>
<b>Contents</b>	<b>iii</b>
<b>I Introduction</b>	<b>1</b>
<b>1 Introduction</b>	<b>2</b>
1.1 Efficiency of the search process . . . . .	2
1.2 Architecture . . . . .	5
1.3 Agent components . . . . .	5
1.4 Cooperative problem solving . . . . .	6
1.4.1 Task sharing . . . . .	7
1.4.2 Result sharing . . . . .	8
1.5 Coordination . . . . .	8
1.5.1 Coordination mechanisms . . . . .	9
1.5.2 Classification . . . . .	11
1.6 Solution direction . . . . .	11
<b>II Research aim</b>	<b>14</b>
<b>2 Research aim</b>	<b>15</b>
2.1 Problem description . . . . .	15
2.2 Research aims and research questions . . . . .	16
<b>3 Methods</b>	<b>17</b>
3.1 Methods . . . . .	17
3.2 Middleware . . . . .	18
<b>III Results</b>	<b>20</b>
<b>4 Architectural drivers</b>	<b>21</b>
4.1 Business attributes . . . . .	21
4.2 Functional attributes . . . . .	21
4.3 Quality attributes . . . . .	21
4.4 Conclusion . . . . .	24

<b>5</b>	<b>Coordination mechanism selection</b>	<b>25</b>
5.1	Coordination mechanisms for task sharing . . . . .	25
5.1.1	Criteria . . . . .	25
5.1.2	Selection . . . . .	28
5.2	Coordination models for result sharing . . . . .	29
5.2.1	Criteria . . . . .	29
5.2.2	Selection . . . . .	30
5.3	Conclusion . . . . .	30
<b>6</b>	<b>Architecture design</b>	<b>32</b>
6.1	Tactic selection . . . . .	32
6.2	Pattern selection . . . . .	34
6.3	Conclusion . . . . .	37
<b>7</b>	<b>Architecture description</b>	<b>38</b>
7.1	Documentation overview . . . . .	38
7.2	Module view search tasks . . . . .	42
7.3	Process view search tasks . . . . .	49
7.4	Module view result sharing . . . . .	52
7.5	Shared data view result sharing . . . . .	56
7.6	Deployment view . . . . .	60
7.7	Mapping between views . . . . .	62
7.8	Conclusion . . . . .	63
<b>8</b>	<b>Architecture evaluation</b>	<b>64</b>
8.1	Prototype . . . . .	64
8.2	Evaluation results . . . . .	64
8.3	Conclusion . . . . .	66
<b>IV</b>	<b>Discussion</b>	<b>68</b>
<b>9</b>	<b>Conclusion</b>	<b>69</b>
<b>10</b>	<b>Discussion</b>	<b>70</b>
<b>11</b>	<b>Future work</b>	<b>73</b>
<b>V</b>	<b>Bibliography</b>	<b>75</b>
	<b>Glossary</b>	<b>81</b>
<b>VI</b>	<b>Appendix</b>	<b>84</b>
<b>A</b>	<b>Quality Scenarios</b>	<b>85</b>
	<b>Samenvatting</b>	<b>90</b>

**Part I**

**Introduction**

# Chapter 1

## Introduction

Searching for radio signals in the ether, , indicated by the term *search*, is considered a complex matter. Moreover, signal searching with the current approaches and resources has turned out to be rather inefficient. In order to improve the efficiency, we aim at deploying autonomous processes. Within the intended situation, these autonomous processes, called agent components, are able to use individual systems to perform search tasks both autonomously and in cooperation with users. In this thesis, we investigate ways in which individual systems can cooperate in order to improve the efficiency of the search process. Task sharing as well as result sharing are (generic) techniques that can be used to solve problems requiring a collective effort. However, both techniques create dependencies between activities. These dependencies have to be coordinated. In this thesis we design an architecture for a distributed system that enables both autonomous and efficient execution of search tasks and sharing of results between these systems and users.

The organisation of this thesis is as follows: In addition to this introduction, Chapter 1 provides the essential background information this thesis is based on. Part II, starting with Chapter 2, describes the main research aims and the research questions that were used to reach these aims. Chapter 3 describes the methods used. Part III, which comprises Chapter 4-8, presents the results: Architectural drivers are described in Chapter 4. In Chapter 5, coordination mechanism and model selection are discussed. The architecture design, documentation and evaluation are described in Chapters 6-8. Part IV, consisting of Chapters 9-11, comprises the main conclusions, the discussion, and future work, respectively.

### 1.1 Efficiency of the search process

This section starts with an explanation of search process background notions used. Next, the complexity of the search process and the signal systems used in this process are discussed. While discussing these items, issues rendering the search process inefficient will be emphasized. The final part of this

section describes the project designed to improve the efficiency of the search process and clarifies the contribution of this thesis to the project.

### Background notions in the search process

In the electromagnetic spectrum several different electromagnetic waves are distinguished which can be expressed in Hertz (Hz). Radio waves (hereafter: signals) can be found in the frequency band from 9 kHz - 3000 GHz. In the scope of this research, the term spectrum refers to signals in the frequency band from 3 MHz - 30 MHz. This band is also known as High Frequency (HF). Transmission systems are used to transfer data between transmitter and receiver. Each transmission system has its own (recognizable) characteristics.

Each signal contains a measurable amount of energy. Detectable signals (energy  $> 0$ ) that are above a certain threshold are referred to as *emissions*. Within an emission characteristics can be found that may match with (known) transmission systems. Finding these characteristics is called *recognition*. *Searching* (for signals) refers to the detection of emissions within the spectrum, if necessary, based on characteristics of transmission systems.

*Search tasks* for signal searching can vary from very simple to very complex. This depends on the characteristics on which the search is based and the frequency band in which the search is performed. An example of a simple search task is: 'make a list with detections on frequency  $f_1$  with antenna  $a_1$ '. An example of a complex task is: 'make a list of all emissions that occur from date  $d_1$  to date  $d_2$  in frequency band  $f_1 - f_2$ , with (or without) the characteristics  $\{c_1, c_2, c_3\}$  or  $\{c_4\}$ , and with antenna  $a_1$ '. The different operations needed for executing of a search task are specified by the term *search process*.

A *filter* refers to a frequency which characteristics are known and is not considered to be of interest. Each frequency within the spectrum may act as a filter. Whenever a search task contains a frequency that matches a filter, this frequency will be neglected.

### Complexity in the search process

The search for signals within the spectrum is considered very complex. This complexity is generated by (i) the size of the spectrum in which a search for emissions is performed, (ii) the number of emissions that can coexist and (iii) the amount of time needed to recognize and track an emission.

It is a fact that multiple emissions (and therefore, transmission systems) can coexist at any moment in time. A-priori, it is unknown whether these emissions are close to- or further away from each other within the spectrum. In addition, due to lack of resources (humans and systems) it is impossible to search the whole spectrum within a relative short period of time, and therefore not all emissions can be detected. Generally, whenever an emission is detected, recognition of this emission is preferred. A user would aim for recognition of the emission by visualizing and/or listening to signals. A software program would do this with support of specific algorithms. In

addition to the fact that manual or automated recognition of emission takes time, one often waits until emission have disappeared. In the meantime no other emissions can be detected. Due to the complexity mentioned above, a lot of emissions remain undetected if only one search task is performed per frequency band. Redundant and overlapping search tasks are assigned to multiple users and/or systems in order to cover as many frequencies as possible.

### Signal systems

For signal searching, several different systems, applications and tools are used. Altogether these are referred to by the term *signal system*. Signal systems differ by the way emissions are detected and recognized. In addition they differ in capacity. The most signal systems do not contain mechanisms for direct integration with other signal systems. Due to the complexity of the search process, it often occurs that similar or overlapping search tasks are executed. To cover an as large as possible part of the spectrum, it is desired to deploy as many as possible signal systems. Nowadays, due to the lack of integration possibilities, knowledge acquired on the basis of the search process is only partially captured and manual tasks to relatively large extent needed for signal searching. Ultimately, this resulted in the current situation where the search process is mainly performed manually.

Due to the fact that the current signal systems lack mechanisms to coordinate redundant or overlapping search tasks, it can not be prevented that a recognition is performed on the same frequency with the same antenna at the same time. Here, emphasis is on the antenna, because it may be useful to use results from the same emission but from different antennas. Altogether, it can be stated that the current search process is rather inefficient.

### Efficiency

Signal searching with the current resources has turned out to be labour-intensive, inflexible and rather inefficient. In order to increase the efficiency we aim at deploying automatic processes. Within the intended situation, these automatic processes are able to perform search tasks both autonomously and in cooperation with users. Moreover, these processes allow signal systems to participate actively in the search process. Furthermore, for optimization of the search process, reuse of knowledge obtained by connecting results from the search process with results from the signal handling process is pursued. We defined a project called *Sirius*. This project aims at developing a prototype of a distributed system that ultimately demonstrates ways to improve the efficiency of the search process. Considering the size of the project, a progressive scheme has been made. Three phases can be distinguished:

1. The development of the architecture of a distributed component system, aiming at independent performance of search tasks by processes

and subsequent communication about these tasks with other signal systems.

2. The expansion of the system with options to control specific signal systems based on the results of the search process.
3. The optimization of the search process by having components reusing knowledge obtained by connecting results from the search process with results from the signal handling process.

The available time for a graduation project was limited and therefore performing all phases of this project was considered to be an unrealistic option. However, this thesis has delivered a substantial contribution to the first phase of this project. The terms architecture and distributed component system are explained in the following Sections 1.2 and 1.3.

### 1.2 Architecture

The term 'architecture' has many faces. First, Rosser [37] distinguish the aspect of time, like a blueprint for the desired situation or a collection of concrete guidelines for future developments. Second, different objects of architecture are distinguished such as the information architecture or technical architecture. Finally, several levels of abstraction are recognized.

The IEEE standard 1471 [28] focusses on systems in which software plays a substantial role in the design, execution and evolution. This standard defines architecture as: "*the fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution*". This definition distinguishes architecture as a concept of a system from the description of an architecture with one or more views. A view is a collection of models that represents one aspect of an entire system from the perspective of a related set of stakeholder concerns.

For this thesis it is important that *(i)* it is about a blueprint of the desired system, *(ii)* there are principles and/or guidelines that direct the development of the architecture and *(iii)* a distinction is made between architecture and an architectural description.

### 1.3 Agent components

As stated in Section 1.1, the signal systems used in the search process are stand-alone systems. In order to improve the search process, it is required that these systems are able to interact with each other. However, these systems were not developed for this purpose and most of these systems do not contain mechanisms for direct integration with other signal systems. One possible solution, as described by Genesereth et al. [20], is to wrap these systems, thereby providing them with an agent layer functionality

and enabling them to communicate and cooperate with other software components.

Heineman et al. [23] define a software component as ‘*a unit of independent deployment that interacts with its environment through its well-defined interfaces while encapsulating its implementation*’. A software component consists of one or more classes and expects an underlying component model. The rationale for the development and implementation of software components is reuse in different software systems. Component based design facilitates the construction of applications by supporting the composition of (simple) components to complex systems. The notion of agent technology, due to the different meanings in literature, often results in different interpretations. Griss et al. [22] consider agents as ‘*next-generation components*’ and agent oriented software development as an extension on component based software development. An agent is often defined as proactive, reactive and social<sup>1</sup>. An agent with such characteristics is goal-oriented, reacts adequately on a changing environment and communicates by means of an agent communication language. So, the term agent technology in the first place refers to the development of software components with additional agent characteristics. Secondly, the analysis and modelling of systems that contain entities with a certain degree of autonomy and activity are meant. In general these entities form a community, which is being considered as an organisation structure. Within this structure entities play one or more roles and interact with each other and their environment.

In this thesis the term agent component (hereafter: agent or component) refers to a software unit, that consists of one or more classes, hides its implementation, uses message oriented communication, takes the initiative to execute a part of a search task or exploits a signal system actively and reacts adequately on changes in the search process. These characteristics give components a limited degree of autonomy that is necessary to perform tasks in the search process independently. These components expect an underlying component model. Agent components are used in the search process to wrap signal systems in agent-like capabilities. A system that consists of this kind of components and that can share results is, in this thesis, referred to as a distributed component system.

## 1.4 Cooperative problem solving

Although agent components support the interaction between stand-alone signal systems, we still need a way to let them work together to perform search process tasks. To put it differently, how can these agent components solve a problem cooperatively. Due to the complexity in the spectrum, cooperation is required for performance of a search task. Techniques originating from artificial intelligence (AI) are often inspired by the way in which people (cooperatively) solve problems. Cooperative Distributed Problem Solving (CDPS or DPS) is an area of AI. With DPS the emphasis

---

<sup>1</sup>We refer to the weak notion of an agent, as described by Wooldridge et al. [47].

lies on the cooperation of individuals to solve problems that require a collective effort.

The main issues with cooperative problem solving (or performing tasks) are addressed by Wooldridge [46]: *(i)* how can a problem be divided into smaller tasks for distribution among components, *(ii)* how can a problem solution be effectively synthesized from sub-problem results, *(iii)* how can the overall problem-solving activities of the components be optimized so as to produce a solution that maximizes the coherence metric, and *(iv)* what techniques can be used to coordinate the activity of components, so avoiding destructive interactions, and maximizing effectiveness. Task and result sharing are techniques that can offer support for these issues. Moreover, these techniques can be used to cooperate individual signal systems.

### 1.4.1 Task sharing

Task sharing can be described as a collection of activities that occur when a search task can not be performed individually or when the execution of a search task be optimized. Durfee [13] distinguishes the following activities of task sharing:

- *Task decomposition*  
Generates the set of tasks to potentially be passed to others. This generally means the decomposition of a search task in subtasks that can be handled by different components.
- *Task allocation*  
The assignment of tasks to appropriate components.
- *Task accomplishment*  
The appropriate components each accomplish their subtask, which could include further decomposition and sub-subtask assignment, recursively to the point that a component can accomplish the task it is handed alone.
- *Result synthesis*  
When a component accomplishes its subtask, it passes the result to the appropriate component, since this one knows the decomposition and thus is most likely to know how to compose the results into an overall solution.

Yang et al. [49] state that implicit dependencies exist between the activities of task sharing and these must be coordinated. Decomposition of search tasks must be performed in such a way that they can be allocated to and handled by components. In addition, tasks have to be defined in such way that their results can be integrated easily. So, task decomposition, allocation, accomplishment and result synthesis must be feasible.

### 1.4.2 Result sharing

Result sharing refers to sharing of information, that is relevant for subtasks of entities, between entities. The main reason for applying result sharing is improvement of group performance. Information can be shared pro-actively (because of the assumption that another component is interested in this information) or can be shared on demand (another component asks for this information explicitly). Durfee [13] described three conditions to improve group performance by sharing information, that apply in this context:

- *Completeness*  
Each component formulates (partial) results for the tasks it can accomplish and these results altogether cover a more complete portion of the overall task.
- *Precision*  
A component needs to know more about the solutions that others have formulated to formulate his own (partial) result.
- *Timeliness*  
Even if a component could in principle solve a large task alone, solving subtasks in parallel can yield an overall solution faster.

Communication matters a lot with result sharing. Sharing of results may have enormous consequences when large amounts of data have to be exchanged and managed. With regard to the search process the sender and receiver are decoupled in space and time. The communication is thus anonymous and asynchronous. The usage of result sharing in the search process requires that the communication is coordinated.

## 1.5 Coordination

Task sharing as well as result sharing are techniques that can be used to make agent components perform search tasks cooperatively. However, both techniques create dependencies between activities. With regard to the activities of task sharing, mutual dependency exists between these activities. With regard to result sharing, the dependency refers to the coordination of communication between components that are decoupled in space and time.

In the search process it is practically impossible to have all entities having knowledge about all search tasks, actions and interactions of all other entities. Therefore redundancy can have a negative impact on the coherent behaviour of the system [24]. From different disciplines, different options, often based on their own definition of coordination, have been described in literature for solving coordination problems.

For example, from the discipline distributed artificial intelligence, Jennings defines coordination as *‘the process by which an agent reasons about its local actions and the (anticipated) actions of others to try and ensure the community acts in a coherent manner’*. He puts forward two arguments to

## 1. INTRODUCTION

---

coordinate actions of multiple components. Considered in the context of the search process:

- *Dependencies exist among actions of components.*  
Dependencies arise, for example, in the case that an emission must be recognized. However, before recognition can take place, an emission must be detected.
- *No single component has sufficient competence, resources or information to solve the problem completely.*  
For example, in case of a search task it is required that one component recognizes the emission whereas another component is required to find the direction of this emission.

Nwana et al. [32] add to this:

- *No single component has a global view.*  
Several search tasks have overlap, but the exchange of information must prevent multiple components to perform a recognition on the same frequency using the same antenna.
- *Information of one component can be used by other components.*  
In case two teams have similarities with regard to their search task, a partial result of one team can, at the same time, serve as a partial result for the other team or just act as a trigger for another group to adjust their search plan temporarily.

A definition that is cited a lot is that of Malone en Crowston [29]. They define coordination as *‘the process of managing dependencies among activities’*. According to them, coordination is only necessary if there is a matter of mutually dependency. In their research, which is also known as *Coordination Theory*, they try to identify which dependencies exist in coordination processes and which generic coordination mechanisms can be used. They investigated coordination processes in different disciplines such as organisation theory, economy and computer science, and identified the following processes: resource and task allocation, goal selection and decomposition and the task/ subtask relation.

### 1.5.1 Coordination mechanisms

In this section mechanisms that can be used to coordinate (some of) the activities of task sharing and the communication with result sharing are investigated.

#### *Organization knowledge*

Organizational structures [29, 42] define the location of control for task decomposition and task execution. Each structure has a dominant form of coordination, which implicitly specifies the activities. Components can use

their organization knowledge to decompose a task, allocate a task and share results.

#### *Planning*

Planning is a mechanism for task decomposition. By means of task networks, each component contains a library of options to realize a plan [15, 31]. This provides a component with the flexibility to anticipate on unexpected failures or unavailable resources. However, the construction of a plan takes time. With multi agent planning the process of creation of a plan as well as the resulting plan itself can be performed centrally or distributed [13]. As a consequence, different combinations are possible to generate a plan. The result, however, always is a plan that can be performed. Such a plan contains the activities, the sequence in which these must be executed and the synchronization points.

#### *Negotiation*

Davis en Smith [10] use the term *negotiate* as a metaphor for task allocation. A coordination mechanism based on tenders is the *Contract-Net Protocol* (CNP) of Smith. Within this mechanisms two roles are distinguished: (i) a manager that decomposes a task in subtasks, searches for contractors that can perform these tasks and supervises the overall solution, and (ii) a contractor that performs a subtask, whereby it is possible that a contractor also takes the role of manager and decomposes the task in subtasks and makes subcontracts with other components. The CNP is a very flexible mechanism for task allocation, accomplishment and result syntheses. Although an auction can also be considered as a mechanism for task allocation, it can be reduced to negotiation with regard to the search process.

#### *Coordination models*

A coordination model can be used to exchange information anonymously and asynchronously as well as to allocate tasks. Examples of such models are the Blackboard model [5], the Linda\* model<sup>2</sup> [19, 35] and the publish/subscribe model [17, 16, 8]. A coordination model can be described with the following elements:

- The coordinated entities that are the subject of coordination.
- The coordination medium is the space where the coordination takes place and were entities are placed in a configuration.
- The coordination laws prescribe in which way the entities are coordinated with the help of the coordination medium.

For some models, reactions can not be adapted, because the semantics is fixed in the model. Reactive models, however, make it possible to program

---

<sup>2</sup> Coordination models that are derived from or contain an extension to the Linda coordination model are being referred to in literature as Linda\* or Linda-Like models.

reactions, that are coupled to specific communication events. This is referred to as a *programmable coordination medium* [11]. Transferring data into or out of a shared memory can be seen as an event, whereupon reactions, programmed in the coordination medium, are executed. Coordination models thus differ in the way in which coordination is defined, of what exactly is coordinated, and in which coordination is achieved [35].

### *Workflow*

Analogous to workflow systems [43, 40], a plan can be defined in advance and fixed for a component. Even though a component is released from planning a task, a dependency is introduced at the same time with respect to the availability of resources. This can be reduced by using agents that represent resources [25, 48]. The dependency between tasks and subtasks can be coordinated centrally within a team, by giving the control over the entire process to a single component.

### 1.5.2 Classification

Schumacher [33] classifies coordination as objective and subjective coordination. Objective coordination is the management of the inter-component dependencies. An example of objective coordination is a coordination model. Subjective coordination is the management of the intra-component dependencies. Subjective coordination can be divided in explicit and implicit coordination. Coordination mechanisms such as organization knowledge, planning, workflow, and negotiation are examples of explicit subjective coordination, because the component treats the management of subjective dependencies internally. Subjective coordination is dependent and based on objective coordination. Because, in this thesis, architecture considers the external properties of components only, subjective coordination mechanisms are not fully shown in an architectural view.

## 1.6 Solution direction

The main task to improve the efficiency of the search process can be subdivided in two partial tasks (*i*) the cooperative and autonomous execution of search tasks and (*ii*) the sharing of results within a team and with users and applications. Task sharing and result sharing are general techniques that can be deployed herewith.

In order to prevent redundant activities plan abstractions can be used. In the partial global planning (PGP) approach of Durfee et al. [14] partial plans are exchanged with other agents in an environment where no global view is available, no global control is possible and the data is distributed. They share partial plans to integrate the partial local interpretations into a coherent overall view. Sharing plan abstractions is adopted in the search process to prevent redundant activities. This is achieved by exchanging partial results. A partial result represents only that part of a plan (e.g. frequency and antenna), that is currently worked on. Other agents can

anticipate on this since they have explicit knowledge of their task structure. In addition, with regard to communication efficiency, exchange of partial results is limited to only those components that have explicit knowledge of the search task.

Cabri et al. [7] describe a case study in which mobile agents<sup>3</sup> on the authority of a user visit a HTML page and analyse this on keywords. If a page contains a keyword and contains cross-links to other pages, these pages are also visited. To prevent several mobile agents to visit the same page, a sign is left behind in a coordination model, that can be read by other agents. Although they use it to exchange information with the same structure between homogeneous agents, we will use this kind of model between heterogeneous agents that use different kinds of information structures.

Task sharing is applied by components to execute search tasks cooperatively and autonomously. The idea is that a manager can allocate a search task to one component. The search task by itself is executed by a team of components. The different activities of task sharing are performed by the team, to achieve independent task performance by the team. A team is generated based on the capacities and resources that are necessary for the search task. Because of this the composition and size can vary per search task. A component that represents a signal system but is not deployed for a search task, will itself take the initiative to join an active team, thereby maximally utilizing resources. The fact that a team of components performing a single search task is, geographically considered, centralised enables a global view and avoids redundancy within the team. However, obtaining a global view between teams and between teams and users is impossible. Result sharing can be adapted here to increase the efficiency between various teams and users.

The communication between components, users and signal systems, with regard to result sharing, runs via markers in a shared memory. A *marker* is being defined here as a frequency on which a filter is applied or on which a component or signal system has detected an emission or on which a recognition is performed. Markers are shared pro-actively. A marker can be a trigger for components or users to change their search task plan temporarily. Components or signal systems that are interested in markers in a certain frequency band can subscribe for this.

---

<sup>3</sup> A mobile agent is a piece of software that exhibits mobility characteristics, through the fact that they can travel over networks and can resume their execution on different systems.



## Part II

# Research aim

## Chapter 2

# Research aim

### 2.1 Problem description

Signal searching with the current resources has turned out to be labour-intensive, inflexible and rather inefficient. We aim for increase of efficiency by computerizing the tasks within the search process. Within the scope of this thesis this means that independent components can form a team of components for a search task and can execute this task, without user intervention. Because of the complexity in the search process redundant and/or overlapping search tasks are assigned to multiple users and/or systems. With regard to efficiency this means that it must be prevented as much as possible that a team, several teams and users are performing a recognition on the same frequency with the same antenna or on a frequency on which a filter is applied.

In Section 1.6 (Solution direction) a global solution is proposed to increase the efficiency of the search process by applying task sharing and result sharing. Task sharing and result sharing implicitly bring along coordination problems. These problems concern the coordination of dependencies between the different activities of task sharing on the one hand and the coordination of communication with result sharing on the other hand. In publications from different disciplines (e.g. organisation theory, economy, distributed systems, artificial intelligence, workflow and grid systems) options for solving coordination problems in environments that correspond to the search process are described. The mechanisms for solving these problems are described in Section 1.5.1 (Coordination mechanisms). These mechanisms are often based on discipline's own definition of coordination and supplied with their own rationale. In literature reports from several studies can be found in which coordination mechanisms are compared to each other for an activity of task sharing or for the communication of result sharing. A comparison based on quality requirements with regard to task sharing and result sharing has not been found in literature so far.

## 2.2 Research aims and research questions

**Aims** are (i) the design of an architecture for a distributed component system in which teams of components can perform tasks from the search process autonomously and efficiently and wherein information can be exchanged mutually between teams and between teams, signal systems and users, and (ii) a Proof of Concept of the designed architecture.

In order to realize this goal the following research questions (RQ) and sub questions (SQ) are formulated:

**RQ 1 How are task sharing and result sharing realized in the search process?**

SQ 1.1 Which are the criteria to evaluate the coordination mechanisms for task and result sharing in the search process?

SQ 1.2 How are the coordination mechanisms for the different activities of task sharing evaluated in the light of the specified criteria?

SQ 1.3 How are the coordination models evaluated in the light of the specified criteria?

**RQ 2 Which roles and interaction models are discerned for task and result sharing?**

SQ 2.1 How are architectural and organisational patterns evaluated in the light of the specified criteria?

SQ 2.2 Which skills are necessary for the execution of tasks in the search process and for sharing results?

SQ 2.3 Which interaction can be derived from the coordination mechanisms and models from RQ 1?

**RQ 3 Which is a proper architectural description?**

SQ 3.1 Which are the appropriate views?

SQ 3.2 Which models are distinguished for these views?

SQ 3.3 Which design decisions apply to more views?

# Chapter 3

## Methods

This chapter describes the methods that were used to realize the results. In addition, the middleware used for the proof of concept is briefly described.

### 3.1 Methods

We have used the Gaia methodology [50] as a guide to develop prototypes with agent components. Gaia is one of the best known and most used agent-orient software development methodologies [21]. However, only the analysis and upper-design phases are addressed.

#### **Architectural drivers**

Due to the fact that Gaia does not directly deal with the activities of (early) requirements engineering [50], we have used the QUINT model [44] to specify quality requirements. We prioritized all (business, functional and non-functional) requirements, which resulted in the architectural drivers. We have used general quality attribute scenarios, defined by Bass et al. [3], to generate concrete scenarios for the quality requirements.

#### **Coordination mechanism and model selection**

For the selection of coordination mechanisms, we performed a literature survey<sup>4</sup>. We investigated coordination mechanisms that are applied in different disciplines for their potential to be used as coordination mechanism for task sharing and result sharing in a multi agent system. In order to compare the mechanisms selected with this survey, we have defined further selection criteria based on the specified quality requirements.

#### **Architecture design**

The architectural design phase in Gaia consists of selecting an organization structure and defining the roles and interaction models within this structure. Due to the fact that architectural structures, other than organization structures, are not excluded in Gaia, we have used the architectural patterns, defined by Buschmann et al. [6]. We selected a collection of tactics [3] to

---

<sup>4</sup>The literature survey was performed as part of the Capita Selecta Graduation.

address the quality requirements. Similar to the *attribute driven design* method [1], these tactics were used to select architectural patterns with which we designed the system.

### **Architecture description**

In order to document the architecture, we have looked at SEI's textbook *Views and Beyond* [9], IEEE 1471 [28], and the Kruchten 4+1 model [26]. Although all three approaches produce an architecture document that consists of a set of views that satisfy the concerns of the architectures key stakeholders and could, therefore, be used, we made the decision to use the *Views and Beyond* (V&B) approach. We have documented the primary structures (selected in the design phase) as views based on the key stakeholders/concerns. Although we have not described all elements of the V&B template, the roles and interaction models have been documented within the views.

Due to the fact that Gaia does not directly deal with particular modelling techniques [50], it is free to use other notations like UML [18]. We have used a combination of graphical notations (which look like UML models) and textual descriptions.

### **Architecture evaluation**

The architecture has been evaluated at several distinct moments in time during the design process. Evaluation was performed by the developers, focussing on the quality aspects. Prototypes, simulation, and scenarios were used. Interoperability and modifiability were evaluated with scenarios during prototype development, whereas reliability, efficiency and portability were evaluated with scenarios by simulating them using the prototypes. In addition, the prototypes have been used to evaluate the architecture with regard to functional requirements.

## **3.2 Middleware**

For the development of prototypes we have used Jade [4] (Java Agent Development Framework) and TuCSoN [34] (Tuple Centre Spread Over the Network) middleware. Jade is a FIPA specifications compliant agent development environment that provides several facilities for an easy and fast implementation [30]. The results of the Gaia design process are agent roles and interaction models. The agent roles are implemented in Jade by specifying behaviour in a Java class and interaction models are implemented with FIPA interaction protocols. TuCSoN supports the communication as well as the coordination of (active) components. We have used tuProlog [12] to specify reactive behaviour in TuCSoN.



**Part III**

**Results**

## Chapter 4

# Architectural drivers

In this chapter only those requirements are specified that have a profound influence on the design of the software architecture. The requirements consisting of a collection of business, functional, and quality attributes in particular are called architectural drivers. The context of the system is presented in Figure 4.1.

### 4.1 Business attributes

The purpose of the system is *(i)* to carry out search tasks efficiently and *(ii)* to share markers between different systems. Because of already existing systems in the search domain that are used for detection, recognition of emissions, and direction finding, use and reuse of these system parts as components is dictated. The system will be introduced with only basic functionality, particularly the part that is responsible for executing search tasks. It must be taken into account that a lot of features are released later on.

### 4.2 Functional attributes

With regard to search tasks, users must have the ability to create, remove, and show search tasks. With regard to result sharing, users must have the ability to add, remove, and show filters. In addition, markers have to be shared between components that are decoupled in space and time.

### 4.3 Quality attributes

The quality attributes are grouped according to the QUINT model [44]. The QUINT model is an extension to the ISO 9126 model for software quality. For each quality (sub)attribute specified, a system specific scenario is generated which can be found in appendix A. These scenarios are derived from general quality attribute scenario defined by Bass et al. [3].

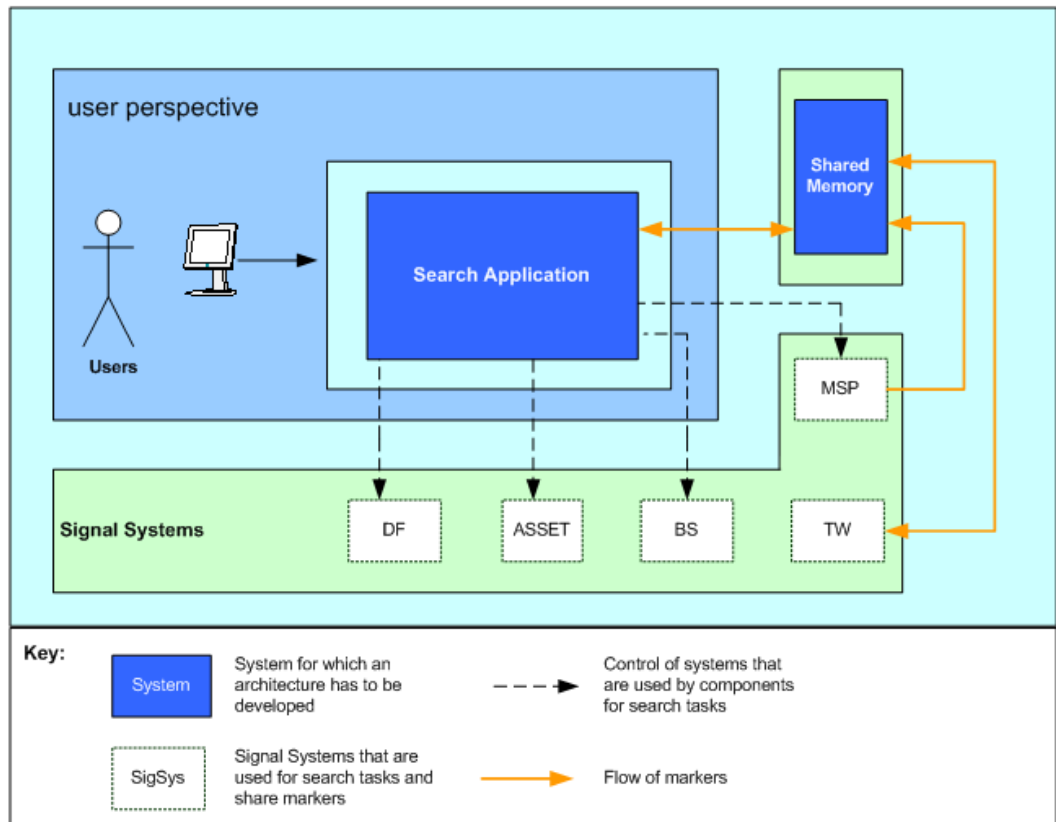


Figure 4.1: Context of the system. This figure only shows those signal systems that are actually used for the prototypes. As a result, other potential usable signal systems are excluded from this figure. In addition, the ASSET and BS systems can only share markers via the Search application.

## Functionality

*This characteristic defines a set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs<sup>5</sup>. The sub attribute that is specified here is interoperability.*

Components in the search process must use signal systems that expose services like detection, recognizing, and direction finding to carry out search tasks. These systems communicate with specific communication protocols such as RMI-IIOP, ATP and Telnet. Because of difficulties associated with integration of different protocols in signal systems it is required that the system to be designed can handle these protocols. Other protocols must be taken into account. The same is true for the shared memory. Signal systems can share their (partial) results via the shared memory and use their own protocol and their own structure to represent a result. The shared memory must be able to handle at least the RMI-IIOP and the FIPA-ACL protocols.

<sup>5</sup>The italicized portions of Section 4.3 are taken from the QUINT model [44].

### **Reliability**

*This characteristic defines a set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.* The two sub attributes that are specified here are recoverability and degradability.

Search tasks are carried out on several locations. One of the problems that is likely to occur is the loss of connection. Although the period of time needed to re-establish the connection is not known on forehand, the system must be able to reconstruct the original connections between components when the connection is available again.

The signal systems that are used to carry out a search task can not be controlled completely. It may occur that (parts of) signal systems cease. Under these circumstances the system must be able to continue the search task with the remaining (parts of) signal systems.

### **Efficiency**

*This characteristic defines a set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions.* The sub attributes time behaviour and resource behaviour are specified here.

Search tasks can be carried out faster if signal systems are optimally utilized. A signal system which is not performing a search task must be detected and initiative must be taken to deploy this system for an existing search task.

In order to guarantee the global coherence, it must be prevented that multiple resources are performing activities on the same frequency with the same antenna. This means that plan adjustments must be possible within each search task.

Search tasks are performed on multiple locations. For the exchange of partial results bandwidth limitations must be taken into account.

### **Maintainability**

*This characteristic defines a set of attributes that bear on the effort needed to make specified modifications.* The sub attribute that is specified here is changeability.

There are multiple signal systems that use their own structure to represent (partial) results. Results of several signal systems differ in the kind of information they present. In addition, these structures can change over time, for example, when new information is added to the structure. Results from signal systems must be exchanged with the shared memory, while preserving their own structure of result representation. New or changed result structures must be modifiable without affecting other systems that exchange results with the shared memory. In addition, a signal system, new to the shared memory, may want to publish or subscribe to markers.

It is expected that there will be a lot of variation in signal systems. These variations may concern (i) the way signal systems are controlled, (ii) new capabilities that are available within an existing signal system, and (iii) a new signal system that needs to be added. This means that it must be possible to simply add or change components without changing other components.

### **Portability**

*This characteristic defines a set of attributes that bear on the ability of software to be transferred from one environment to another.* The attribute that is specified here is adaptability.

Some (parts of) signal systems are currently running on a Windows or Linux platform. Although the user interface runs on a Windows client machine, the same components must be able to run on a Windows as well as a Linux platform without changing the software and without losing functionality.

It is expected that the system is expanded gradually in order to search a larger part of the spectrum. The system needs to be horizontally scaleable in order to support this growth.

## **4.4 Conclusion**

In this chapter we have specified those requirements that have a profound influence on the design of the architecture. For each quality (sub)attribute specified, a system specific scenario was generated (see Appendix A).

The specified requirements in this chapter are all requirements with the highest priority with regard to the defined requirements plan (which is not defined in this public document). However, we consider efficiency (*performance*), although important to guarantee the global coherence, less essential than the other quality requirements defined in this chapter.

## Chapter 5

# Coordination mechanism selection

In this chapter, the selection of coordination mechanisms and a coordination model is described for task sharing and result sharing, respectively. First, the criteria that distinguish the mechanisms and models are specified. Second, a selection is made. The selection of mechanisms for task sharing is described in the first section. The next section describes the selection of a model for result sharing.

### 5.1 Coordination mechanisms for task sharing

When comparing mechanisms for task sharing, it is important to emphasize that not all mechanisms can be used for all activities of task sharing. Moreover, for each activity of task sharing, separate comparison and selection of the coordination mechanisms is necessary. Table 5.1 shows which mechanisms are compared for each activity of task sharing.

#### 5.1.1 Criteria

For the selection of coordination mechanisms it is important to know the type of task (from routine to innovative) and the environment wherein the task is executed (from relatively stable to very unpredictable).

Within the search process the type of task is considered as relative routine. With regard to a search task two decompositions are distinguished. First, a global decomposition (detecting, recognizing, and direction finding) is made based on the required system used for detection, which is specified by the user. Second, the decomposition of a frequency list is needed to schedule detections. The first is known at forehand and the latter can only be defined during execution. As a result, the main decomposition of a search task is very stable, which makes the creation of a plan less complicated. The environment is considered relatively stable. Although the search tasks are executed in a closed environment, and all entities are designed to cooperate, the environment is unpredictable with regard to loss of connections or cease of (parts of) signal systems.

## 5.1. COORDINATION MECHANISMS FOR TASK SHARING

---

	Decomposition	Allocation	Accomplishment
Organization knowledge	●	●	
Task networks	●		
Workflow	●		●
Negotiation		●	
Coordination models		●	●
Multi agent planning			●
<i>Covered</i>	●		

Table 5.1: Coordination mechanisms for task sharing.

For the selection of mechanisms, the quality attributes (specified in Section 4.3) must be considered. Not all defined attributes have an equal profound influence on an activity of task sharing and thus on the selection of a coordination mechanisms for this activity.

The attributes portability and interoperability do not influence the selection of coordination mechanisms for task sharing. With regard to efficiency, a search plan needs continuous adjustment during plan execution. Moreover, a global view is necessary within a search task for the exchange of partial results. These affect the task accomplishment activity. In addition, communication must be taken into account since it affects the task allocation and accomplishment activities. The scalability affects the task allocation activity. With regard to modifiability, capabilities of signal systems can change. As a result, the task decomposition and task allocation can change. With regard to reliability, connections can be lost or (parts of) signal systems can cease. This affects the task allocation and task accomplishment activities.

We believe that the coordination mechanisms for the activities of task sharing can be compared based on the characteristics mentioned above. Highlighted are those criteria that are considered dominant for mechanism selection for one or more activities of task sharing.

- *Adjustable*  
A coordination mechanism for task accomplishments is considered adjustable if parts of the search plan can be changed during execution.
- *Efficient*  
A coordination mechanism for task allocation and accomplishment is considered efficient if as few as possible communication is necessary.

## 5. COORDINATION MECHANISM SELECTION

---

- *Scaleable*  
A coordination mechanism for task allocation is considered scaleable if the addition of one or more signal system components is transparent for the allocation mechanism.
- *Changeable*  
A coordination mechanism for task decomposition is considered changeable if the change of a capability affects one component only.
- *Adaptable*  
A coordination mechanism for task allocation is considered adaptable if components have a low coupling in order to anticipate on changing capabilities of signal systems.
- *Reliable*  
A coordination mechanism for task accomplishment is considered reliable if it can recover the connections between components after a connection break down.
- *Stable*  
A coordination mechanism for task allocation is considered stable if tasks, in case of caesing of (parts of) signal systems, can still be allocated to other (parts of) signal systems of the same type.
- *Centralized*  
A coordination mechanism for task accomplishment is considered centralized if a global view within a search task is available to prevent the execution of redundant subtasks.

The above-described criteria show their relation with one or more activities of task sharing. Since coordination mechanisms can be used for one or more of these activities (see Table 5.1), it is possible to use these criteria to compare the mechanisms. Table 5.2 shows the scores of the coordination mechanisms for these criteria. For each coordination mechanism, only those criteria are scored that are associated with the task sharing activities that can indeed be performed by this mechanism. For each task sharing activity, the mechanisms and associated score are discussed below.

### **Task decomposition**

Task decomposition using organization knowledge or a workflow plan is efficient with regard to communication, because the plan structure is known at forehand. With task networks, communication must take place before a plan can be created. Organizational knowledge is considered to be unsuitable for change, because it will affect more than one component since the components must have knowledge of each other. Task networks and workflow plans score better, because the change can be made at single point. Overall, the workflow plans score best for task decomposition.

	Adjustable	Efficient	Scaleable	Changeable	Adaptable	Reliable	Stable	Centralized
Organization knowledge		●		○				
Task networks		◐		◐				
Workflow	○	●		◐		●		●
Negotiation		◐	●		●		●	
Coordination models	●	●	●		●	○	●	○
Multi agent planning	●	◐	◐		○	◐	◐	◐
<i>Good</i>	●							
<i>Sufficient</i>	◐							
<i>Bad</i>	○							

Table 5.2: Coordination mechanisms for task sharing set out against selection criteria.

### Task allocation

For task allocation, negotiation is initially communication-intensive, due to the agreement that has to be made. Negotiation as well as coordination models have a low coupling between components and are able to assign tasks in case (parts of) signal systems cease. The mechanism organization knowledge defines a high coupling between components and as a result is less stable in case signal systems cease. In addition, it is less scaleable than the other mechanisms. Coordination models seem to score best here, although negotiation scores good as well.

### Task accomplishment

For task accomplishment, coordination models are considered less suitable because a global view of the search task is lacking. Moreover, reconstructing connections is not possible due to the space and time decoupling of components. A workflow mechanism is not usable with regard to adjustments, that must be made during plan execution, but scores very well on other characteristics. This is in contrast with multi agent planning, which is highly adjustable, but scores less on other criteria. Multi agent planning with centralized planning for distributed execution seems to score best, when all criteria are considered.

#### 5.1.2 Selection

As Yang et al. [49] stated, implicit dependencies exist between the activities of task sharing. The mechanisms used for the different activities must fit together. For task sharing, the following mechanisms are selected:

(i) for task decomposition a workflow plan is used, (ii) for task allocation negotiation (CNet protocol) is used, (iii) for task accomplishment workflow and multi agent planning mechanisms are used. Result synthesis, which highly depends on the mechanism for task accomplishment, is realized by the CNet protocol and workflow.

## 5.2 Coordination models for result sharing

### 5.2.1 Criteria

The basic functionality for result sharing is to provide communication between components that are decoupled in space and time. Moreover, the model must support publishing and subscription services for components. In addition, the quality attributes interoperability and maintainability (specified in Section 4.3) for result sharing must be considered for the selection of a coordination model.

With regard to interoperability, the shared memory must be able to handle systems that use different protocols to publish markers and/or subscribe for markers. With regard to modifiability, markers from different systems, that use their own specific marker representation, can change over time. More specifically, the structure representing marker information changes. Changes in a structure of one system may not affect structures used by other systems. In addition, these changes can be supported by the components itself or centrally with a programmable coordination medium.

We believe that the coordination models for result sharing can be compared based on the characteristics mentioned above. Highlighted are those criteria that are considered dominant for model selection.

- *Decoupling*  
A coordination model is considered decoupled if it supports space and time decoupling between components that communicate via the model.
- *Interoperable*  
A coordination model is considered interoperable if it can support multiple communication protocols.
- *Changeable*  
A coordination model is considered changeable if a change in an information structure does not affect other components.
- *Programmable*  
A coordination model is considered programmable if reactions can be specified within the coordination medium.

The comparison for result sharing (Table 5.3) shows that all models directly support time and space decoupling between components. The

	Decoupling	Interoperable	Changeable	Programmable
Reactive Linda*	●	◐	●	●
Blackboard	●	◐	○	◐
Publish/Subscribe	●	◐	◐	◐
<i>Good</i>	●			
<i>Sufficient</i>	◐			
<i>Bad</i>	○			

Table 5.3: Coordination models for task sharing set out against quality criteria.

usage of different protocols highly depends on the specific model implementation. In addition, support for different protocols can be realized using brokers. Only the reactive Linda\* model is considered fully supportive to changes in the information structure without a ripple effect, but a programmable coordination medium is required. With regard to a blackboard or publish/subscribe model another component is necessary to relate the different structures. A publish/subscribe implementation in which the coordination can be programmed is not excluded. Only reactive Linda\* models support a programmable coordination medium.

### 5.2.2 Selection

With regard to the coordination models evaluated, the choice is made for a reactive Linda\* coordination model. The reactions are specified in the coordination medium. More specific, the different structures used by the systems are mapped in the medium.

## 5.3 Conclusion

This chapter provides the answers to research question RQ 1: *How are task sharing and result sharing realized in the search process?* Although the mechanisms selected are suited for the activities of task sharing and communication between components for result sharing in the search process, the criteria are far too specific to use them as general criteria for selection in other domains.

Based on the classification of Schumacher [33] it can be stated that subjective coordination is used for task sharing. That is: the coordination for task sharing is handled explicitly within the agents. Because architecture

## 5. COORDINATION MECHANISM SELECTION

---

in this thesis only considers the external properties of components, these mechanisms are not fully shown in an architectural view. With regard to result sharing, objective coordination is used. This model is shown explicitly in an architectural view (see Chapter 7 Architecture description).

The coordination mechanisms for task sharing and coordination model for result sharing, which are selected in this chapter, can be considered as design decisions. The next chapter describes the fundamental design decisions that shape the architecture.

## Chapter 6

# Architecture design

This chapter describes the fundamental design decisions that were made and the pattern selections performed to realize these decisions. The first section describes which tactics were selected and how they can be achieved. Bachmann et al. [2] define an architectural tactic as *a means of controlling a quality attribute measure by manipulating some aspect of a quality attribute model through architectural design decisions*. The next section describes the selection of (a collection of) architectural patterns, to realize the selected tactics. An architectural pattern is defined by Bijlsma et al. [5] as *a proven structural organization schema for software systems*.

### 6.1 Tactic selection

Bass et al. [3] describe a number of tactics that can be used here. The selected tactics are based on the architectural drivers (defined in Chapter 4) and the derived concrete scenarios (defined in Appendix A). The quality attributes described by the QUINT-model [44] do not have a one-to-one mapping to the attributes defined by Bass et al. [3]. As a result interoperability and portability are considered here as part of maintainability.

#### **Maintainability**

It is expected that most changes will occur with regard to signal systems. When the way in which signal systems are controlled changes, it does not only affect the module that uses this system, but also other modules depending on the changed module. This ripple effect can be prevented by *encapsulating implementation details*. *Maintaining existing interfaces* can prevent a change in a capability of a signal system to affect more than the module representing this signal system. The tactics mentioned above also decouple the binding between modules, which is always beneficial with regard to modifiability.

The addition of new signal systems will directly affect other modules. The tactic *abstract common services* can be used to group related services together, in order to localize the change and prevent ripple effects.

## 6. ARCHITECTURE DESIGN

---

With regard to portability, modules are replicated or moved within one location or to other locations. It is necessary that dependent modules know where to find them. To determine the location of a service, an *intermediary* (name server) can be used for registering the service and allowing this location to be discovered.

With regard to result sharing, data producers and consumers need to share markers anonymously and asynchronously in a publish/subscribe manner. In addition, producers and consumers use their own data structure to present markers. A change in this structure will affect all consumers. A (modern) *repository* allows the consumer to specify the structure in which data is presented, regardless of the data's structure in the repository. Although a repository decouples the modules in space and time and breaks down structure dependency, it does not address interoperability issues. An *intermediary* can be used for protocol translations between the signal system and the repository.

### **Efficiency**

It is required that the system reacts within a specific time to markers. In case a resource of another search task is performing the same subtask or a filter is specified for a scheduled frequency, the local search plan is adjusted temporarily for the period of time that another resource is busy with the same subtask or the filter is active. This will guarantee the global coherence. In addition, when a transmission is detected or recognized, a partial result has to be sent to the shared memory. The tactic *manage event rate* can be used here to reduce latency. Reducing the number of modules that have to be informed about a marker event or need to publish a partial result, will reduce latency. In addition, partial results are only exchanged within a location and only if a transmission occurs. To increase the overall performance of search task execution, *concurrency* can be introduced.

### **Reliability**

In order to detect a communication breakdown between locations, a *ping/echo* tactic can be used. A module at the main location sends a ping message to the connected service. If the service does not reply (echo) within a certain amount of time, it is assumed that the communication between these locations is lost.

In case a communication time-out occurs between a module and a signal system, an *exception* is raised in order to detect the signal system being ceased.

Attribute	Tactics used	How Achieved
Maintainability	Encapsulation	Hide signal system control issues from other modules
	Maintain interfaces	Use one generic interface for detector modules and one for recognizer modules
	Abstract services Intermediary	Group signal systems together Name server (pub/sub) for runtime location discovery
Efficiency	Manage event rate	Centralize task management Send partial results only when detected or recognized
	Introduce concurrency	Parallel execution of detector and recognizer tasks
Reliability	Intermediary	Check with name server for service existence
	Exception	Signal system request time-out

Table 6.1: This table shows for each quality attribute which tactics are used for task execution and how they are achieved. The name server is supplied as part of the middleware.

Attribute	Tactics used	How Achieved
Maintainability	Repository	Reactive coordination model
	Intermediary	Bridge to connect different protocols
Efficiency	Manage event rate	Exchange markers only on location

Table 6.2: This table shows for each quality attribute which tactics are used to share markers and how they are achieved.

## 6.2 Pattern selection

A pattern is a set of predefined subsystems and their responsibilities that implements a collection of tactics. Due to the fact that these tactics often concern different quality attributes, the decision for a pattern, in order to realize some tactics, may have an impact on other qualities. In choosing a pattern, these trade-offs have to be considered [3]. Buschmann et al. [6] describe a number of patterns that can be used to implement a number of tactics mentioned in Section 6.1.

### **Task execution module**

With regard to task execution, the primary trade-off is between tactics that improve modifiability and reliability and the tactic to reduce latency in order to improve the performance. The modifiability tactics enable easy integration of new and change of existing signal systems. The use of these signal systems is essential to improve the search process. The tactics to increase modifiability may also increase execution time and may create additional network traffic. In addition, introducing concurrency will have a positive effect on the performance (the number of frequencies that can be covered in a certain amount of time), but introduces additional network traffic which has a negative effect on the latency. Reducing latency is considered an important tactic, because exchanging partial results, within a certain amount of time, can prevent that the same subtask is performed by multiple agents and/or systems. Although this is considered essential with regard to improving the efficiency of the search process, the costs of an occasional double result is taken for granted with regard to the increase of frequencies that can be covered. As a result, we consider performance (reducing latency) less important than modifiability, reliability, and resource behaviour.

For task execution the following patterns are selected:

1. Layers pattern
2. Broker pattern
3. Master-slave pattern

Although the layers pattern as well as the broker pattern implement the modifiability tactics mentioned in Section 6.1, the layers pattern is selected as the primary pattern that provides services for task execution at different levels of abstraction (see Figure 6.1). These services are implemented by virtual machines. Use of virtual machines promotes modifiability and is a mechanism for abstraction [41]. In addition, it engenders portability [9] and breaks down the dependency.

Figure 6.1 shows the user interface at the top layer. Because a user and, therefore the user application are not present all day, the user service layer provides search task services to the user. In addition, this layer has the responsibility to detect communication break downs and connection reconstruction. The two lowest layers are used to implement the master-slave pattern and the lowest layer, where all services are grouped for controlling signal systems, is implemented with the broker pattern.

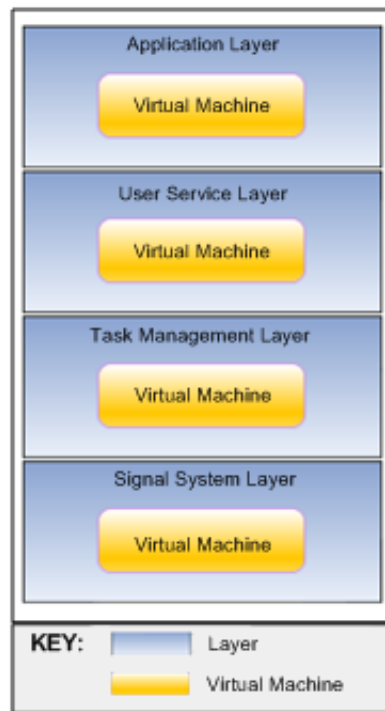


Figure 6.1: Architectural pattern that utilizes tactics to achieve modifiability and reliability tactics for task execution.

### Result sharing module

The tactics for modifiability and efficiency do not have a negative impact on each other. For sharing markers the modifiability tactics for result sharing (see Section 6.1) are implemented by the following patterns:

1. Repository pattern
2. Broker pattern

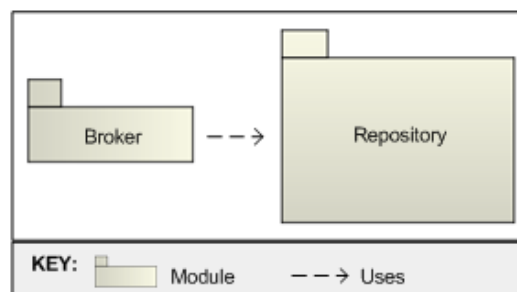


Figure 6.2: Architectural pattern that utilizes tactics to achieve modifiability for result sharing.

The repository pattern (*reactive coordination model*) is used to provide anonymous and asynchronous communication between publishers and

subscribers. Moreover, it supports the mapping of different information structures inside the model, supporting change of these structures. The broker pattern is used to translate a system specific protocol to the protocol used by the coordination model.

### 6.3 Conclusion

In this chapter, the fundamental design decisions have been made. First, a number of tactics have been selected based on the quality attributes (maintainability, performance and reliability). Second, two module structures (search task and result sharing module) have been decomposed based on these tactics. A trade-off has been made between modifiability, reliability and performance. This chapter answers a part of the second central question SQ 2.1: *How are architectural and organisational patterns evaluated in the light of the specified criteria?* With regard to the search task module, the layers pattern has been selected as the primary structure and within the broker and master-slave pattern. With regard to the result sharing module the repository and the broker pattern have been selected. In Chapter 7 (Architecture description), functionality is allocated to these structures.

# Chapter 7

## Architecture description

In this chapter functionality is allocated to the primary structures that have been selected in the previous chapter. Moreover, this chapter describes the designed architecture with various views. A view is a collection of models that represents one aspect of an entire system from the perspective of a related set of stakeholder concerns [28]. The first section provides information about how the documentation is organized, what the architecture encompasses, and why the architecture is the way it is. The other sections document the views.

### 7.1 Documentation overview

#### 7.1.1 Stakeholders

The following stakeholders are considered to be of primary importance: customer/acquirer, developers, testers, and users. The customer is a fictitious organisation. Developers are *(i)* the members of the development team, *(ii)* designers of other systems with which the current system has to interoperate, and *(iii)* maintainers that need to perform maintenance activities. The testers are members of the development team and/or designers of other systems that need to test specific behaviour. The users are people that use this system to generate search tasks, create filters, and view search task results. Table 7.1 shows which architectural views are considered to be important for these stakeholders.

#### 7.1.2 Documentation roadmap

This section provides a brief description of the views presented in Sections 7.2 - 7.6. Selection of views was based on the stakeholders (and their concerns) described in Table 7.1.

##### **Module view search tasks** (see Section 7.2)

This view shows how the software for task execution is structured as a set of four layers. The highest layer provides task services to the user and the lowest layer provides abstract signal system services. The intermediate

## 7. ARCHITECTURE DESCRIPTION

	Search tasks		Result sharing		
	<i>Module view</i>	<i>Process view</i>	<i>Module view</i>	<i>Shared data view</i>	<i>Deployment view</i>
Customer/acquirer	○		○		○
Developer	●	●	●	●	○
Tester	◐	●	◐	●	◐
User		○		○	○
<i>All info</i>	●				
<i>Some info</i>	◐				
<i>Overview info</i>	○				

Table 7.1: Stakeholders and their relevant views.

layers provide abstract task services for the upper layers. In addition, this view shows the allocation of functionality to agents and the interactions between agents with regard to task allocation. Developers and testers can find additional information with regard to modifiability and portability.

### **Process view search tasks** (see Section 7.3)

This view represents the system as a set of concurrently executing agents. It shows the interaction between agents when performing multiple search tasks and the protocols they use to communicate. Detailed information is provided with regard to how a search task is planned and executed and about performance. In addition, this view shows agent reactions to emission detections and partial results of other agents or systems.

### **Module view result sharing** (see Section 7.4)

This view shows how the system for result sharing is decomposed into a repository and broker modules. The repository provides anonymous and asynchronous communication between brokers that provide publish/subscribe functionality for systems that want to share markers. Detailed information is provided with regard to modifiability.

### **Shared data view result sharing** (see Section 7.5)

This view shows the result sharing systems as a set of running modules. It shows the interaction between systems and agents when sharing markers. Moreover, it shows how the repository handles marker exchanges between publishers and subscribers internally. Detailed information is provided with regard to modifiability and performance.

### **Deployment view** (see Section 7.6)

This view shows how (agent) components and repositories are deployed to nodes and their relation with middleware platforms. Moreover, it shows how these nodes are distributed with regard to locations. In addition, it provides performance and reliability information.

### 7.1.3 View template

The architecture documentation templates from Clements et al. [9] were used as a guide<sup>6</sup> to describe the architecture. The views are described using the next elements:

#### **Primary presentation**

The primary presentation is a graphical presentation showing the elements and relationships among them.

#### **Element catalog**

The element catalog details the elements depicted in the primary presentation. For each element in the primary view the properties, relations, interfaces and behaviour are described.

#### **Context diagram**

This diagram shows how the system relates to its environment.

#### **Architecture background**

The background describes the rationale for the view and the assumptions that were made.

### 7.1.4 System overview

The main purpose of the system is to provide autonomous execution of search tasks and information sharing between entities. The system consist of two (independent) system parts: one for search task execution and one for result sharing (see Figure 4.1 in Chapter 4).

The search system part provides search services to users, who would normally use several individual signal systems in the search process for detection, recognition and direction finding. An user can formulate a search task, in which capabilities of several signal systems can be combined. Moreover, the system can perform the task without user intervention.

The result sharing system part provides exchange of search related information between entities, which are decoupled in space and time, and use this information to increase their search task performance. In addition, it provides users with services to define filters and show task results.

The number of users that will concurrently use the search system part is estimated at three (current practice). However, the number of search tasks that will be concurrently active is not known at forehand. The number of systems sharing results is estimated at four and based on the most important

---

<sup>6</sup>Not all elements prescribed by the templates were implemented or fully described.

systems. The number of concurrent connections with the result sharing system depends on the number of search tasks and the location where the tasks are executed.

### 7.1.5 Rationale

#### Architecture

We think that the efficiency of the search process can be improved by combining capabilities of individual signal systems and by sharing information between these systems (via the agents). In order to achieve this, the architecture must be able and flexible to use these signal systems easily, even if they change. As a result, the architecture is designed to meet maintainability requirements maximally. In order to perform search tasks without user intervention, reliability requirements are met as well. Performance requirements are considered to be less essential.

#### Middleware

Due to the fact that reuse of (parts of) signal systems was dictated, agents were used to exploit these systems. Middleware was used to reduce the development time and to have a standard component model for these agents. The decision for the middleware was based on *(i)* government guidelines to use open source software, *(ii)* offered middleware services, *(iii)* possibility to run the middleware in a Windows as well as on a Linux environment, and *(iv)* available documentation. Although there are several suitable agent frameworks that meet these concerns, the decision was made for Jade. The Jade middleware allows components with their own thread of control. This is important, because connections with other systems require threads within the agent component. For result sharing the decision was made for the TuCSon coordination model. Although JavaSpaces has been considered, it did not provide a coordination medium. The weak point of the TuCSon middleware is the experience in an academic setting only.

#### Communication

For agent communication the decision was made for the FIPA-ACL. This was based on *(i)* government guidelines to use standardized open communication protocols, *(ii)* standard interaction protocols for task assignment, *(iii)* implementation of this protocol by the used Jade middleware, *(iv)* possibility to use different contents (e.g. Java objects and XML), and *(v)* possibility to use this protocol within a C++ or .NET application.

For communication with signal systems we decided to comply with the protocols of these systems, since it is not possible or very difficult for some signal systems to use another communication protocol to communicate with external systems (e.g. search system or result sharing system).

#### System division

In a very early stage we decided to split up the system in two different independent system parts: one for search task execution and one for result

sharing (see Figure 4.1 in Chapter 4). A first reason is that it is easy to deploy one part without the necessity to use the other. For example, users performing signal search with signal visualisation systems can see activities of other users. A second reason is that the two parts can be developed individually. Since the project, for which this architecture is designed, is at a very early stage, separate development of the two parts may ultimately be the case.

### 7.1.6 Other information

The evaluation of the architecture is described in Chapter 8. As a result, evaluation information is not documented in the views.

## 7.2 Module view search tasks

### Primary presentation

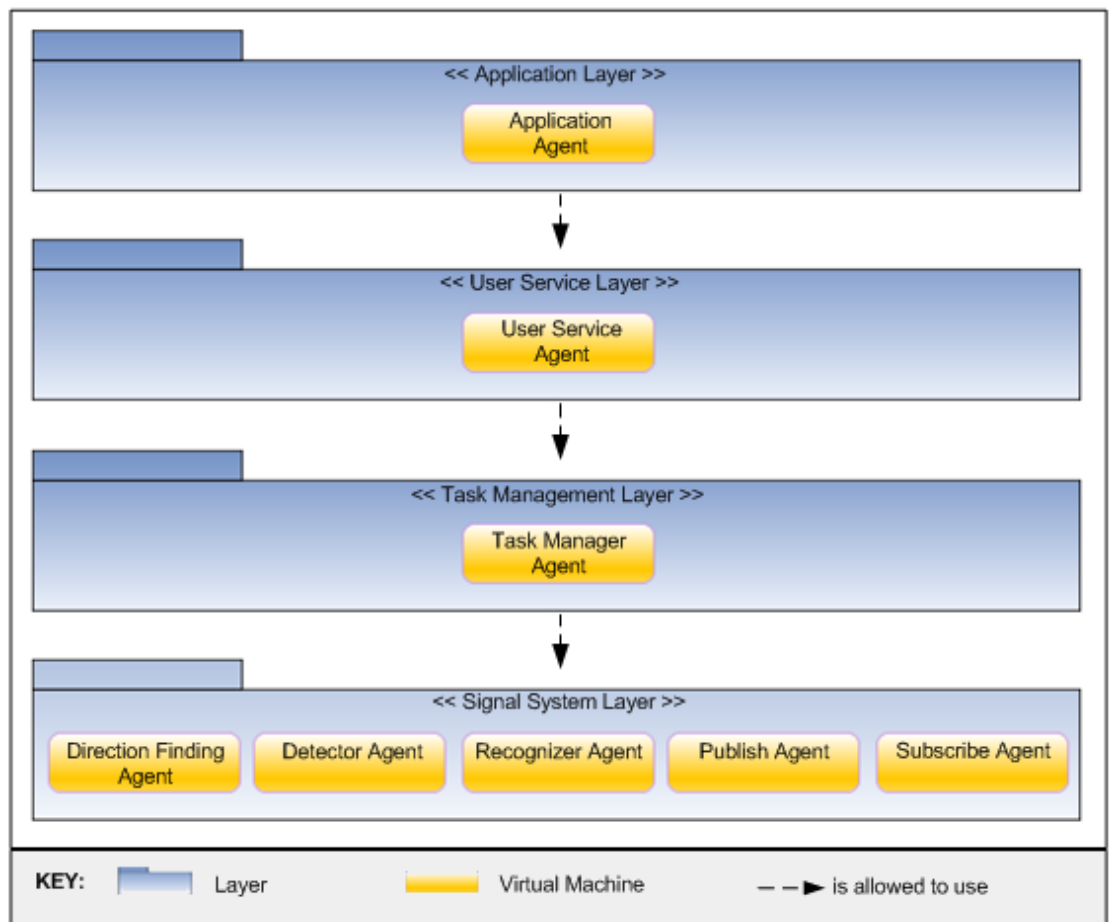


Figure 7.1: Module view search tasks.

## 7. ARCHITECTURE DESCRIPTION

---

### Elements and their properties

Application	Provides task services to users.	Application Agent
User Service	Provides abstract search task services to the application layer and monitors the search task execution.	User Service Agent
Task Management	Provides search task execution for the user service layer.	Task Manager Agent
Signal System	Provides signal system services for the task management layer to perform search task specific tasks.	Detector Agent, Recognizer Agent, Dir. Find. Agent, Publish Agent, Subscribe Agent

Table 7.2: Layer descriptions search tasks.

Element	Responsibilities
Application Agent	Contains an user interface to users for search task requests. It provides services to the user for creating, showing, and removing search tasks. All search task requests are forwarded to the user service agent, which handles the request for the user. Due to the fact that this agent is only available for the time the user is present, it will not save any search task or task related information.
User Service Agent	Provides search task services for application agents. It registers the start and stop date/time for each search task and the task manager agent that actually performs this task. This information is used to <i>(i)</i> schedule a task, <i>(ii)</i> stop a task, and <i>(iii)</i> restore connections with task managers after a connection break-down. It can communicate with other user service agents to create an overview of all search tasks.
Task Manager Agent	Provides search task execution services. This agent creates a central search plan for distributed execution, based on the capabilities provided by individual signal system agents and user requirements. In addition, it synthesizes, formulates, and exchanges partial results and can adjust the local search plan based on actual plans of other agents or systems. Signal system agents that are used to perform a part of the search task are registered for the period of time the task is running.

Element	Responsibilities
Detector Agent	Provides an abstract service for emission detection. The detector agent translates the task requests from the task manager agent to signal system specific requests. Due to the fact that signal systems differ in capability, control and task execution, detector agent implementations are different for each signal system.
Recognizer Agent	Provides an abstract service for emission recognition. The recognizer agent translates the task requests from the task manager agent to signal system specific requests. Like the detector agent, recognizer agent implementations are different for each signal system, the main differences being on-line (streams) and off-line (files) recognition.
Dir. Find. Agent	Provides services to determine the direction of an active emission. It can handle multiple requests from multiple task manager agents. This agent translates the requests to a signal system specific request.
Publish Agent	Provides a service to task manager agents to publish a partial result to the shared memory.
Subscribe Agent	Provides a content-based subscribe service to multiple task managers. The subscription service is valid for partial results as well as for filters created in or removed from the shared memory.
Yellow Pages	Provides a publish/subscribe name server for all agents. The yellow pages is part of the middleware. Due to the fact that the primary presentation only focusses on the search system part and not on the middleware, this element is not shown in the primary presentation.

Table 7.3: Agent descriptions search tasks.

### Relations and their properties

The relation in the layered view is allowed-to-use. Agents in a layer are allowed to use agents in any other layer immediately below. Only agents in the User Service Layer are allowed to use other agents in the same layer. The relations between agents and the yellow pages is not shown in the primary presentation.

### Element interfaces

All agents use the JADE-ACL protocol. In the search task context, two kinds of interfaces must be known in order to use agent services: (i) message envelop to which an agent can respond, and (ii) agent service description in the yellow pages. These are described below. The performative values are defined in the JADE middleware package *jade.lang.acl.ACLMessage* and the other values are defined in the source package *sirius.search\_one.ontology*.

## 7. ARCHITECTURE DESCRIPTION

---

	<b>User Service Agent</b>	<b>Task Manager Agent</b>	<b>Direction Find. Agent</b>
<i>Envelope</i>			
Performative	CNP REQUEST	CNP	CNP
ConversationID	SEARCHTASK CID.STATUS	SEARCHTASK	CID_DF_TASK
Language	JAVA.LANG	JAVA.LANG	JAVA.LANG
Content Object	SearchTask	SearchTask	DfTask
Ontology	SEARCH	SEARCH	SEARCH
<i>Service</i>			
Name	user-service	task-service	sigsys-service
Type	searchtask	searchtask- management	direction- finding

	<b>Detector Agent</b>	<b>Recognizer Agent</b>
<i>Envelope</i>		
Performative	CNP	CNP
ConversationID	CID_DET_TASK	CID_REC_TASK
Language	JAVA.LANG	JAVA.LANG
Content Object	DetectionTask	RecognizerTask
Ontology	SEARCH	SEARCH
<i>Service</i>		
Name	sigsys-service	sigsys-service
Type	detector	recognizer

	<b>Publish Agent</b>	<b>Subscriber Agent</b>
<i>Envelope</i>		
Performative	CNP	CNP
ConversationID	CID_PUB_MARKER	CID_SUB_MARKER
Language	JAVA.LANG	JAVA.LANG
Content Object	SearchTask- PartialResult	Marker- Subscription
Ontology	SEARCH	SEARCH
<i>Service</i>		
Name	sigsys-service	sigsys-service
Type	publisher	subscriber

Table 7.4: Interface description for search task agents.

**Element behaviour**

Agents use the JADE CNP interaction protocol for task assignments (see Table 7.4). The fixed sequence of messages, prescribed by this protocol, and agent role change is shown in Figure 7.2.

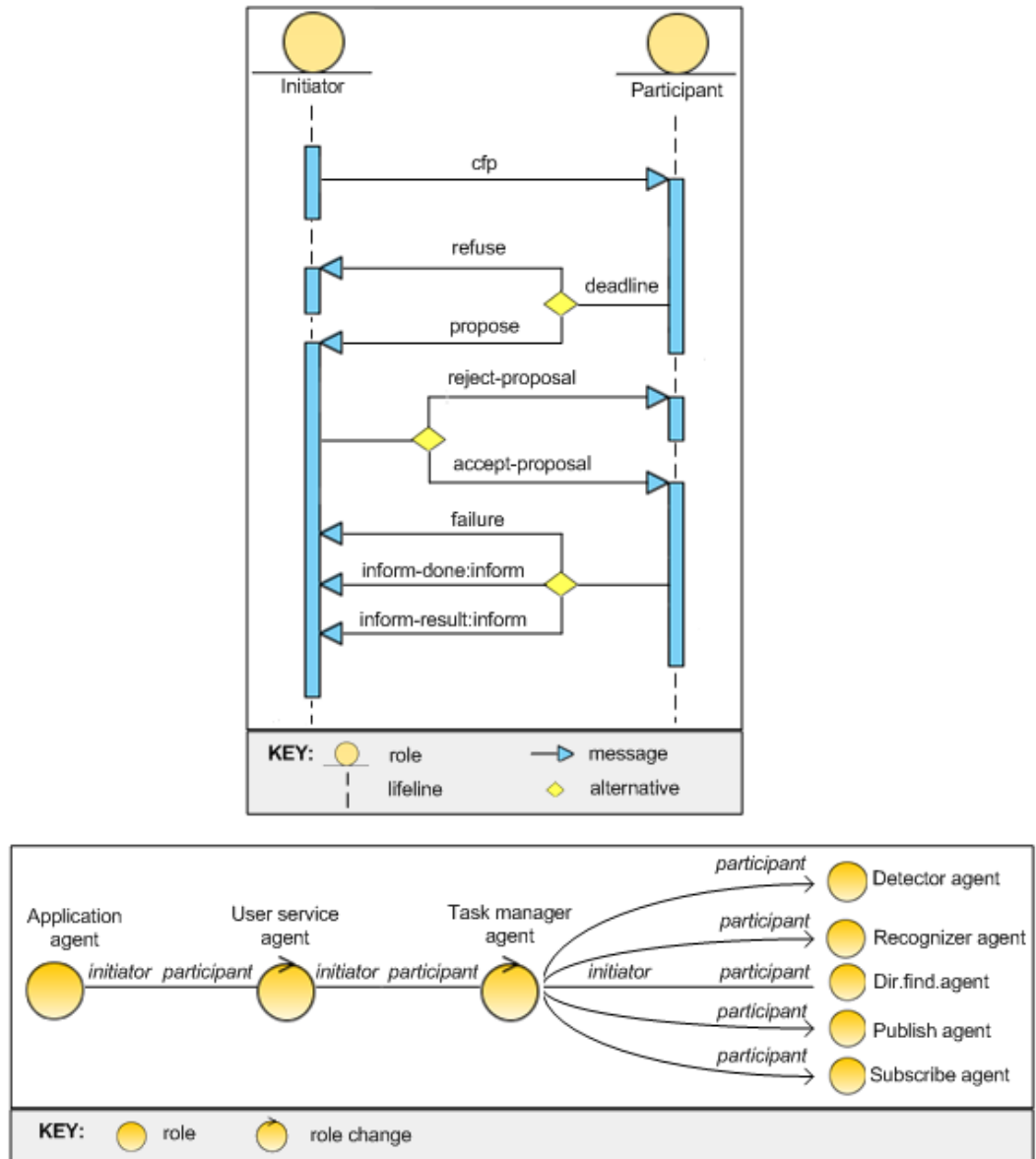


Figure 7.2: Task assignment behaviour.

## Context diagram

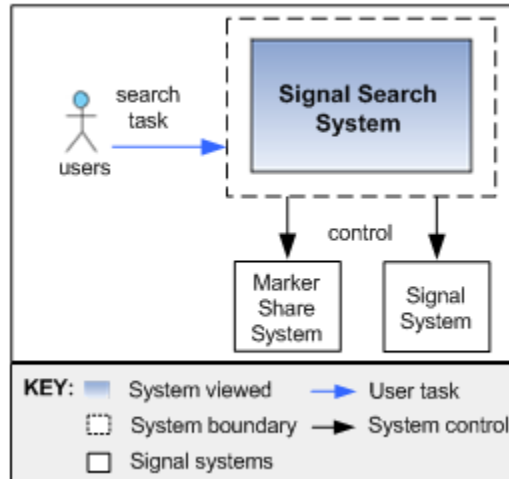


Figure 7.3: Context task sharing.

## Architecture background

## Rationale

The decision for the layers pattern is particularly based on the achievement of maintainability requirements. Grouping all related services together (for example one detector service for all detector agents) will localize changes. The signal system agents are considered as brokers that hide their implementation. Changes in a signal system are kept local inside the agent representing this system and, as a result, will not affect other agents that use the agent's service.

Search tasks are created to run for a relatively long period of time. An user creating a search task, is available for only a short period of time. As a result, the application agent is available for this short period of time as well. Due to the fact that search tasks can be performed at other locations and connections can break down, an overview of all search tasks is unavailable. The user service layer is an additional layer that provides the service to represent the user's search task 24x7 within the system. Moreover, agents within the user service layer can detect communication break downs and reconstruct the connections when the communication is available again.

The use of the yellow page service provided by the middleware, makes it possible to find (new) agents whenever they are replicated or moved within one location or to other locations. In addition, signal system agents that can not provide their service anymore (due to signal system failure, for example) unregister from the yellow page in order to prevent their deployment for search task execution. Signal system agents detect unused signal system capacity and utilize these systems by publishing their service

in the yellow pages. Task manager agents, which subscribe to the yellow pages for signal system services (provided by detector and recognizer agents) receive a notification in case the service is available and will try to assign tasks to the available agent. This is a workaround, due to the fact that the layers pattern, as defined here, does not allow usage of services from the layer above.

In order to achieve modifiability we decided to create abstract interfaces and to keep these interfaces stable. The interfaces defined in Table 7.4 provide the possibility to change, replace or add new components without affecting other components. For example, all detector agents implement the same interface and use the same registration with the yellow pages, despite the different implementations and signal systems they represent.

### **Alternatives**

The blackboard and peer-to-peer patterns were considered as alternatives to the layers pattern. The blackboard pattern enables effective distribution of (partial) tasks between signal system agents. Changes in detector or recognizer agents can be made without affecting other components. One of the problems that might occur is the number of agents performing a task. Due to the fact that search tasks differ in complexity, different search tasks require less or more agents. The blackboard pattern can not directly prevent all agents to contribute to the same search task, while at the same time another search task can not be performed due to the lack of available agents. Moreover, the decoupling between agents makes it difficult to restore connections between agents running on different locations, in case of a connection break down.

The peer-to-peer pattern is often used in agent systems in which agents use services of each other. The requirement to utilize idle signal systems can be easily met if a detector agent can use the services provided by the user service agent or the task manager agent, while at the same time a task manager agent can use the services provided by the detector agent. Although this would be very flexible, removing or replacing agents can affect multiple other agents.

### **Assumptions**

- It is expected that most changes will occur with regard to signal systems.
- At each location a (default) search task runs permanently.

### 7.3 Process view search tasks

#### Primary presentation



Figure 7.4: Process view search tasks. This figure shows a scenario in which one user executes two search tasks and another user executes one search task. In addition, it shows how multiple detector and recognizer agents are used within one search task.

#### Elements and their properties

All agents in the primary presentation are process elements, having their own thread of control. Thread scheduling of these processes is handled by the underlying (JADE) middleware with a round-robin strategy. All tasks performed by an agent (e.g. communicating, detecting, planning, etc.) are handled by individual *behaviours* defined within an agent. As a result, an agent voluntarily suspends its own execution (e.g. detection behaviour) to receive a new task.

### 7.3. PROCESS VIEW SEARCH TASKS

Element	Type	Description
FIPA-ACL	Message passing	Asynchronous protocol. The underlying middleware buffers the messages.
ATP	Message passing	Asynchronous protocol. The messages are buffered.
Telnet	Message passing	Asynchronous protocol. The messages are buffered.
Publish/Subscribe	Data exchange	Repository publish/subscribe mechanism. The underlying middleware provides data buffering and asynchronous operation.

Table 7.5: Connector descriptions search tasks.

#### Relations and their properties

The only relation in this view is *attachment*, dictating how agents and connectors and agents and external signal systems are attached to each other. The relation between agents and the yellow pages is not shown in the primary presentation.

#### Element interfaces

See Section 7.2 for the search task agents and Section 7.4 for the result sharing agents.

#### Element behaviour

See Figure 7.5 and Figure 7.6.

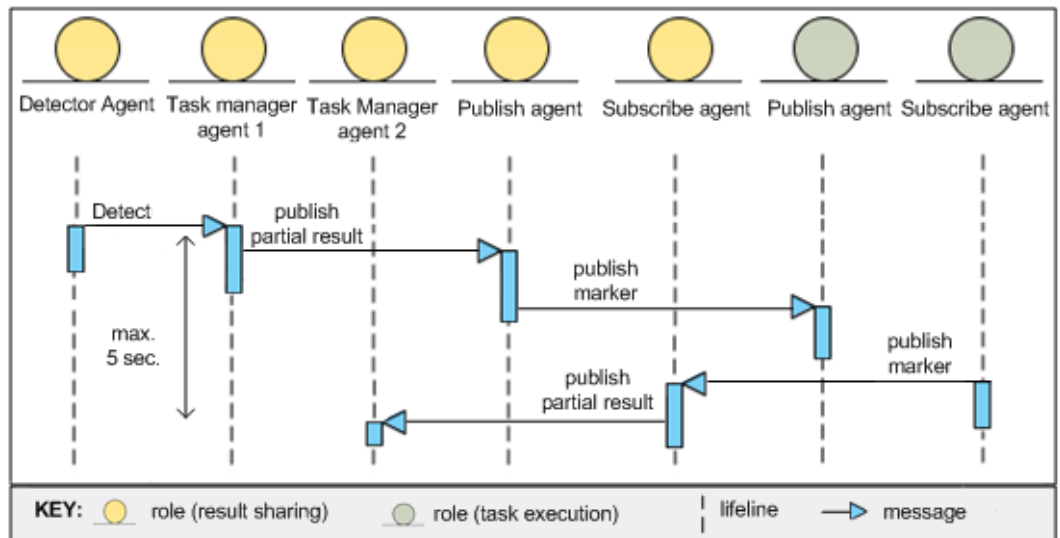


Figure 7.5: Partial result behaviour.

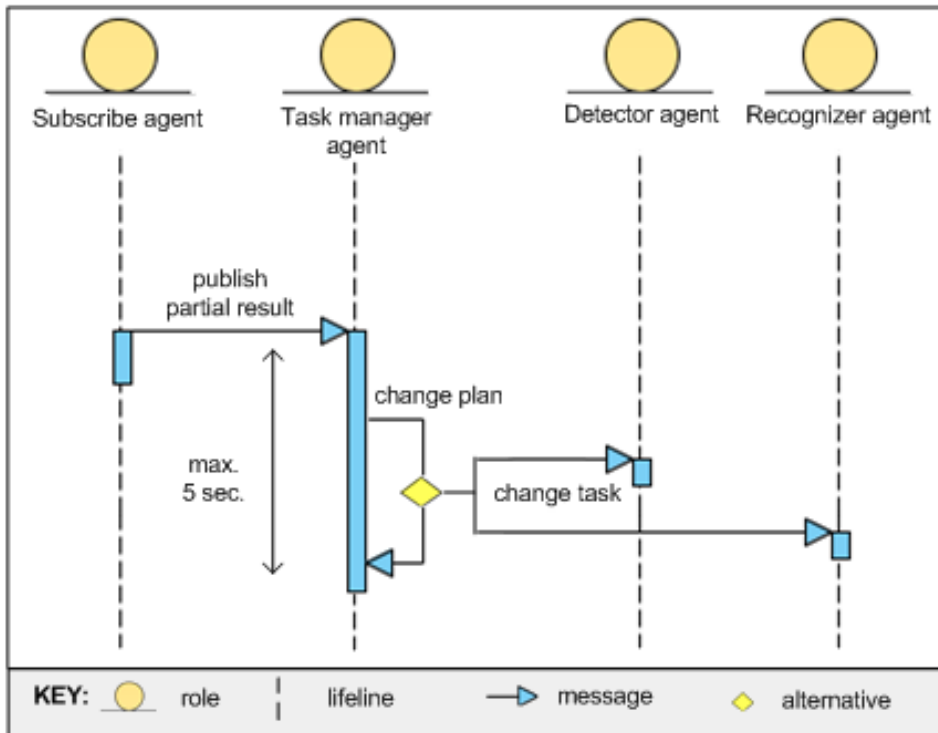


Figure 7.6: Marker reaction behaviour.

### Context diagram

See Figure 7.3 in Section 7.2.

### Architecture background

#### Rationale

The decision to make a task manager agent the key player within the execution of a search task has a lot of performance benefits. The task manager handles almost all task sharing activities. Due to the fact that the task manager knows the complete search plan for a single search task, it can efficiently react to markers created by others, and adjust the search plan. As a result latency is reduced, as all other agents, performing partial tasks, do not have to react to these markers. Moreover, all kinds of partial results, created by the agents performing parts of the search plan, are synthesized by the task manager and published as a single partial result in order to manage the event rate. The master-slave pattern (task manager agent with multiple detector and recognizer agents) is used to increase the overall performance of search task execution. Due to the fact that a task manager has a global view within one search task, it can be prevented that multiple agents perform the same subtask.

### Alternatives

The main alternative is to decentralize task sharing activities. The complete search task is assigned to multiple agents and the execution of this task, the formulation of partial results, as well as reaction to markers have to be performed by every agent working on this task. Although this will make the task manager agent redundant, latency will increase, as all agents must share their partial results and all have to react to markers of all other agents. Moreover, within a search task, execution of the same subtask can not be prevented due to the lack of a global view.

### Assumptions

With the selection of the master-slave pattern, it is assumed that the benefits associated with covering a higher number of frequencies within a certain period of time will outweigh the costs of the increased latency this pattern will bring along.

## 7.4 Module view result sharing

### Primary presentation

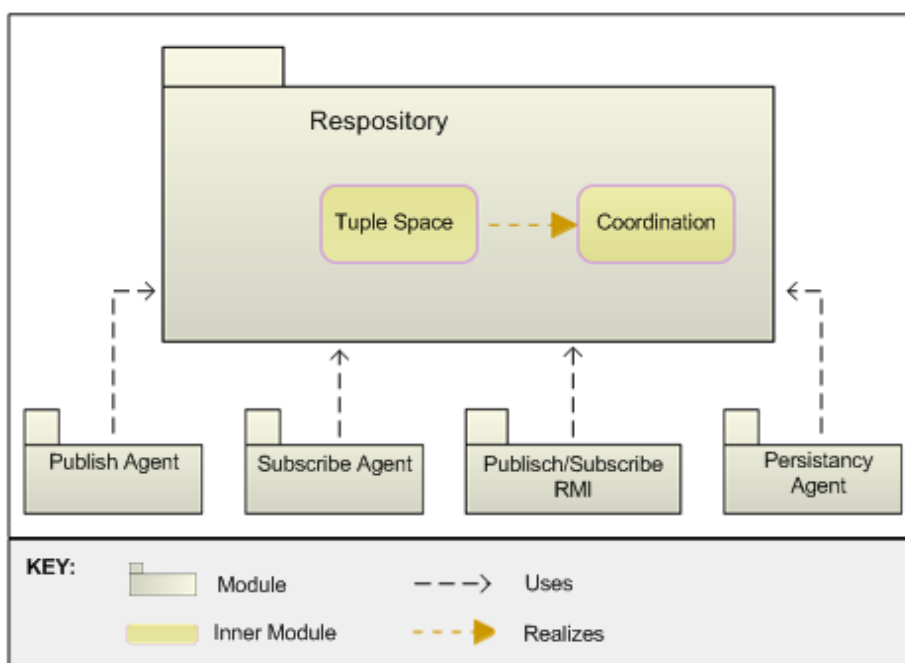


Figure 7.7: Module view result sharing.

### Elements and their properties

Element	Responsibility
Repository	Provides publish/subscribe services for marker exchange. Moreover, it supports anonymous and asynchronous communication between publishers and subscribers. In addition, it provides a coordination medium wherein reactions can be specified to certain data events.
Publish Agent	Provides a bridge between the repository and entities communicating with an ACL, wanting to publish markers (partial results and filters).
Subscribe Agent	Provides a bridge between the repository and entities communicating with an ACL, wanting to subscribe to markers (partial results and filters).
Publish/ Subscribe RMI	Provides a bridge between the repository and entities, communicating with RMI-IIOP, that want to publish or subscribe to markers (decorators).
Persistancy Agent	Subscribes to the repository for markers (partial results and filters) and writes these to persistent storage. Filters are written to one file (serialized object) and partial results are written to files (XML) archived by day.

Table 7.6: Module descriptions result sharing.

### Relations and their properties

The relation type in this view is the *uses* relation. All modules rely on the correct implementation of the repository. However, the publish/subscribe RMI module forms an exception to this relation. This module can provide data exchange to publishers and subscribers, which use this module, even if the repository is unavailable. The relations between agents and the yellow pages is not shown in the primary presentation.

### Element interfaces

With regard to agent interfaces (and their rationale) see Section 7.2. The interfaces used by the publish/subscribe RMI module are described in the source package *sirius.share\_one.tuplespace.rmi*. The operations for data exchange with the repository are defined in the TuCSoN middleware package *alice.logictuple*.

Due to the fact that RMI clients wish to receive event notifications from the publish/subscribe RMI component, all clients need to be remote objects themselves. The interfaces are described in Table 7.7.

#### 7.4. MODULE VIEW RESULT SHARING

	Repository (Tuple Space)	Publish/ Subscribe RMI		
Publish data	outp(logic_tuple)	RMIcallbackServerTS		
Subscribe	inp(logic_tuple)	RMIcallbackClientTS		
	Publish Agent	Subscribe Agent	Persistancy Agent	
<b>Envelope</b>				
Performative	CNP	CNP	REQUEST	
ConversationID	CID_PUB_MARKER	CID_SUB_MARKER	CID_FILTERS	CID_RESULTS
Language	JAVA_LANG	JAVA_LANG	JAVA_LANG	
Content Object	SearchTask- PartialResult Filter[]	Marker- Subscription	SearchTask- Result Filter[]	
Ontology	MARKERS	MARKERS	MARKERS	
<b>Service</b>				
Name	data-service	data-service	data-service	
Type	publish	subscribe	persistancy	

Table 7.7: Interface description for result sharing modules.

#### Element behaviour

The behaviour of the modules and the repository is described in Section 7.5.

#### Context diagram

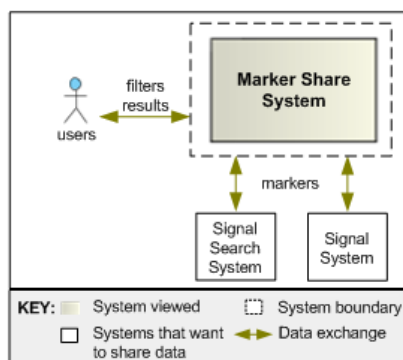


Figure 7.8: Context diagram result sharing.

### Architecture background

#### Rationale

Data producers and consumers need to share markers anonymously and asynchronously in a publish/subscribe manner. The decision for a repository enables the decoupling of systems and users in space and time. Publish/subscribe behaviour is realized by specifying coordination rules within the coordination medium. Moreover, coordination rules are used to map different information structures (see Section 7.5 for more information), in order to prevent a change in one structure to affect all data consumers.

Since we decided to comply with the protocols of existing systems and since not all signal system can use the repository API, brokers are used to bridge the different protocols used.

Another decision made was to combine publish/subscribe behaviour in the publish/subscribe RMI component, independent of the repository. Which means that RMI clients (signal visualisation systems) can exchange markers, without the availability of the repository. The main reasons are *(i)* to have direct functionality available to improve the search process with manual search, and *(ii)* the fact that it is uncertain whether the repository will be available on all locations in the short term.

The persistency agent was added later on, due to the fact that the repository middleware did not provide services to write information to persistent storage.

#### Alternatives

An alternative for the combination of a repository with brokers is to only use a publish/subscribe mechanism. Although this will deliver the basic functionality requested, the change of one information structure will create a ripple effect in more than one producer or consumer.

#### Assumptions

It is expected that the information structures of partial results change over time. Search task results that are written to persistent storage are transferred to the main location by existing file managers. Every location has its own result sharing system (see Section 7.6). It is expected that the repository will be replaced by or connected with a database.

## 7.5 Shared data view result sharing

### Primary presentation

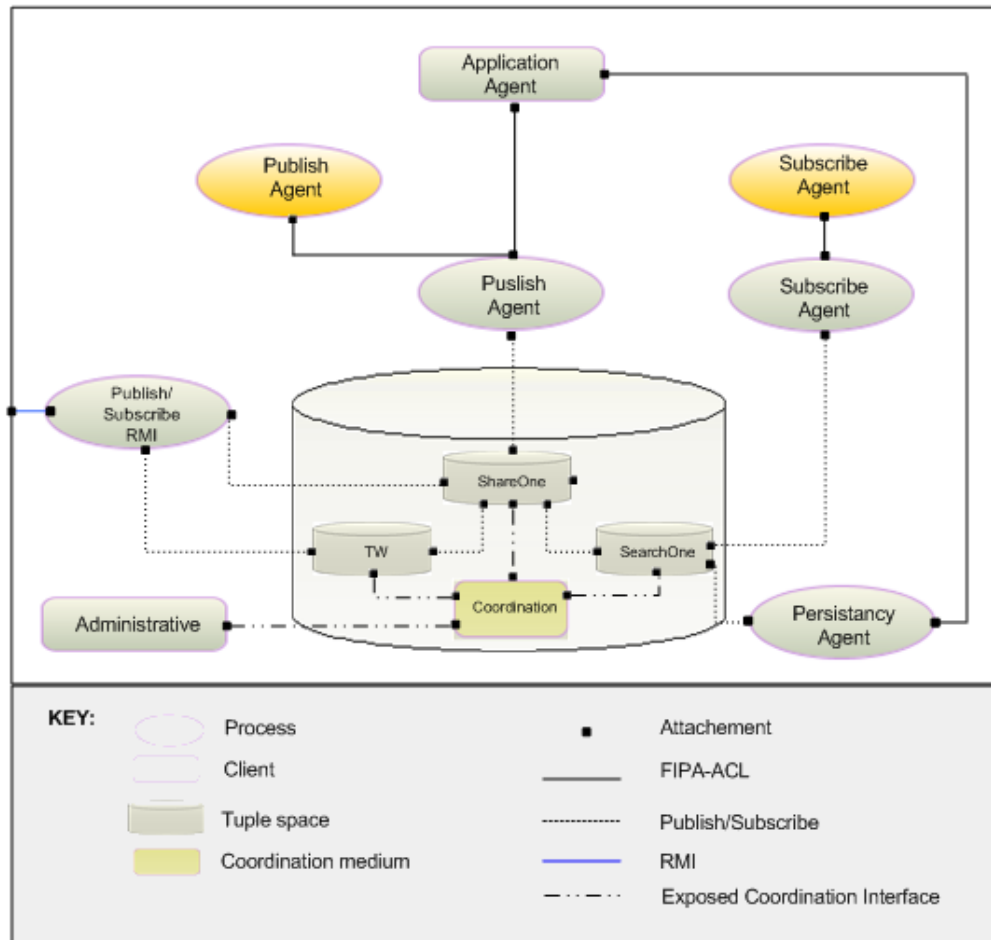


Figure 7.9: Shared data view result sharing.

### Elements and their properties

See Section 7.3 (*Elements and their properties*) for more information about agent properties.

## 7. ARCHITECTURE DESCRIPTION

Element	Type	Description
ShareOne / SearchOne / TW	Repository	The repository (TuCSon tuple space) does not have any restrictions on the number of accessors. In addition, concurrent access is permitted. New accessors can be added at runtime and have unrestricted access to the data in the repository. The data is not persistent. Coordination rules are embedded within the repository specifying how the repository reacts to a publish event. The ShareOne repository is used by all data accessors that publish markers. The coordination rules for the ShareOne repository specify that all data is copied to the SearchOne and TW repositories. Data accessors, that subscribe to events, create their own tuple space (at run time) in which the repository can copy new events. The coordination rules for the SearchOne and TW repositories specify that all data is copied to the personal tuple spaces, based on the subscription information of the subscribers.
Publish Agent	Data accessor	Publishes markers (partial results or filters) to the ShareOne repository. New events can be created dynamically.
Subscribe Agent	Data accessor	Subscribes to marker events (partial results or filters) announced by the SearchOne repository. Subscription to new events can be created dynamically.
Publish/ Subscribe RMI	Data accessor	Publishes markers (decorators) to the ShareOne repository. Subscribes to marker events (decorators) announced by the TW repository. New events and subscription to new events can be created dynamically.
Persistancy Agent	Data accessor	Subscribes to marker events (partial results or filters) announced by the SearchOne repository. Subscription to new events can be created dynamically.
Application Agent	Client	Provides an user with services to create, show and remove filters and to show search task results in the repository.
Administrative	Client	Provides an user with services to specify the coordination rules for each tuple space.
FIPA-ACL	Message passing	Asynchronous protocol. The underlying middleware buffers the messages.
Publish/ Subscribe	Data exchange	Repository publish/subscribe mechanism. The underlying middleware provides data buffering and asynchronous operation.
RMI	Message passing	Synchronous protocol. The messages are not buffered.
Coordination interface	Protocol	Specifies coordination rules within a repository.

Table 7.8: Connector descriptions search tasks.

### Relations and their properties

The relation of this view is attachment, dictating how clients, publishers, subscribers and repositories are attached to each other. Besides the omitted relation between the agents and the yellow pages, there are no additional relations other than the ones shown in the primary presentation.

### Element interfaces

See Section 7.4.

### Element behaviour

See Figure 7.10 for publish/subscribe behaviour of the elements.

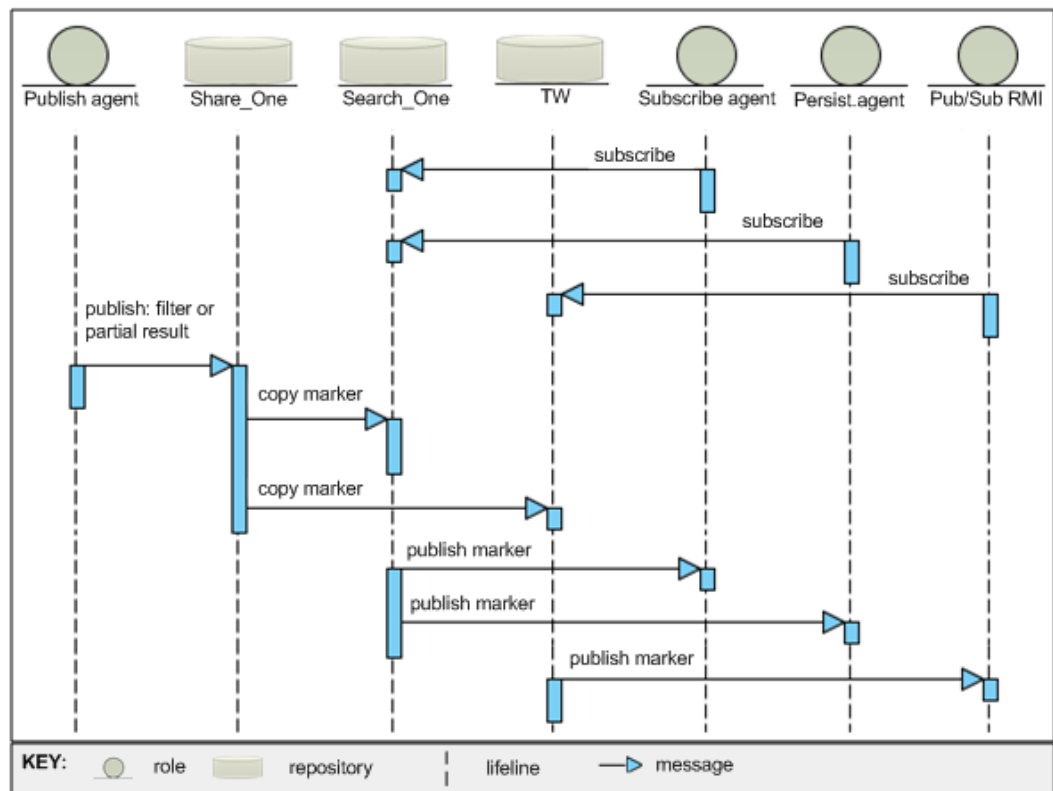


Figure 7.10: Publish/subscribe behaviour.

### Context diagram

See Figure 7.8 in Section 7.4.

### Architecture background

#### Rationale

Due to problems encountered with a single tuple space for publishing and subscribing (see Section 8.2 for more information), the decision was made to split up the tuple space in one tuple space for publishing and one tuple space for each type of consumer using a specific information structure.

Within the publish tuple space, coordination rules are specified in order to map and copy the published data to the subscribing tuple spaces. The advantage of specifying these coordination rules in the coordination medium is that a change is localized only to the broker that uses this structure and the repository mapping this structure. In addition, producers and consumers of data only require knowledge of their own information structure. As a result, other data consumers are not affected by this change. The same is true if new signal systems, with their own information structure, are added to the shared memory system.

The disadvantage of using these coordination rules is that, in order to share markers with systems new to the repository, coordination rules must be defined. Moreover, these rules must be defined for each tuple space used.

#### Alternatives

Use one repository for publishers and subscribers.

#### Assumptions

The assumptions are similar to those of Section 7.4.

## 7.6 Deployment view

### Primary presentation

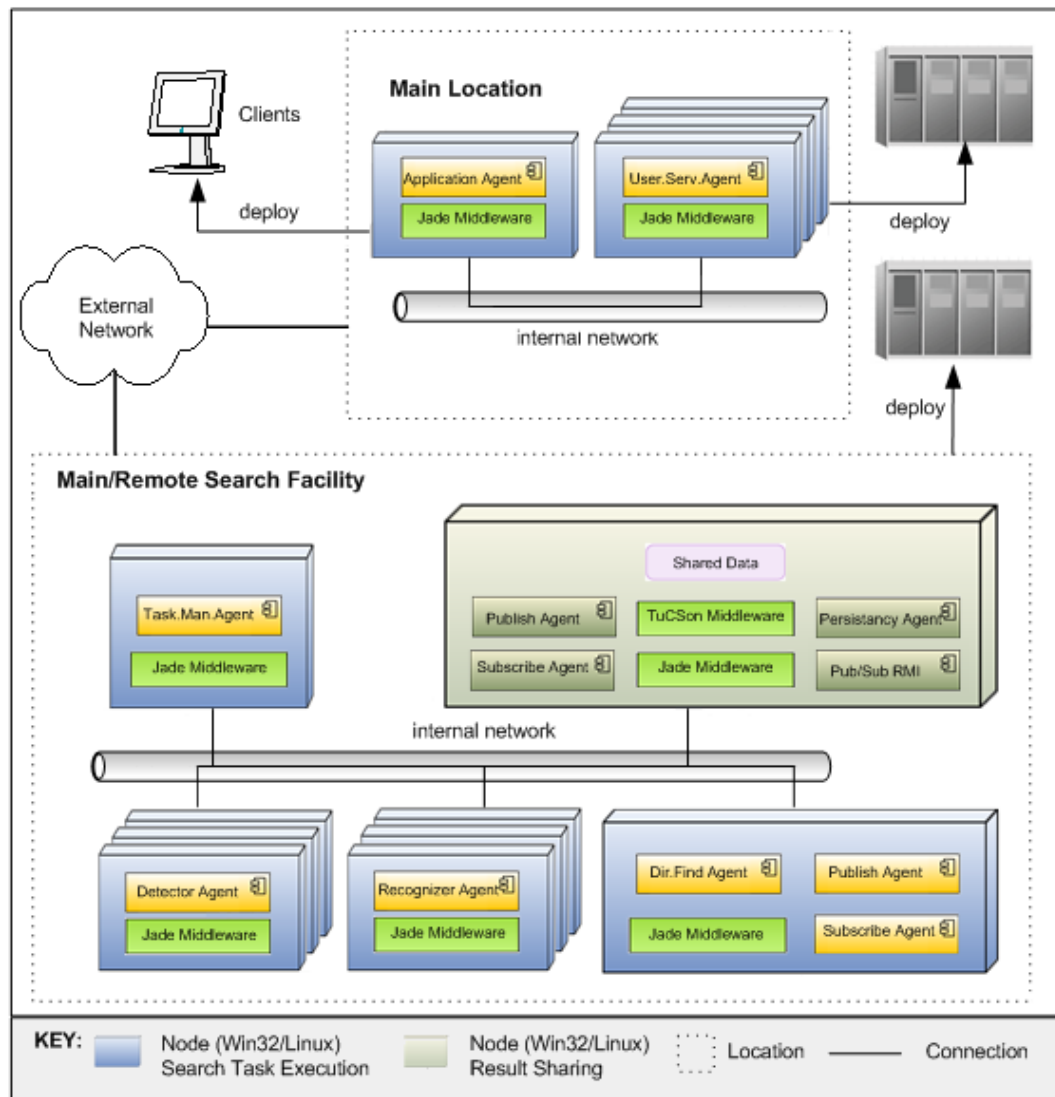


Figure 7.11: Deployment view.

## 7. ARCHITECTURE DESCRIPTION

---

### Elements and their properties

Element	Description
Node	The nodes can be Windows 2000/XP/Vista or Linux RH 6.x (or newer) computers. In order to run the middleware properly, Java JRE 1.4.2 (or newer) must be installed.
Jade Middle-ware	This middleware is the environment wherein agent components run. The services that are used from this middleware are: <i>(i)</i> agent communication service, <i>(ii)</i> yellow page (publish/subscribe) service, and <i>(iii)</i> fault tolerant platform service. A Jade platform consists of one main container and multiple subcontainers distributed over multiple nodes.
TuCSon Mid-dleware	This middleware provides a tuple center, which can contain multiple tuple spaces. Reactive behaviour is programmed in the coordination medium.

Table 7.9: Element descriptions deployment view.

### Relations and their properties

The primary presentation shows the allocated and is-allocated-to relation. The relations are as shown in the primary presentation.

### Element interfaces

Not applicable.

### Element behaviour

Not applicable

### Context diagram

See Figure 4.1.

### Architecture background

#### Rationale

Due to the fact that connections between (remote) locations are not always guaranteed and to reduce network traffic between locations, the result sharing system is deployed at each remote location. As a result, sharing of markers is only performed at, instead of between, locations.

In Section 7.3 the decision was made to use the master-slave pattern (one task manager agent and multiple detector and recognizer agents for one search task). In order to reduce network traffic between locations and to provide reliability, task manager agents are deployed on remote locations. As a result, a search task that needs to use detector and recognizer agents

on a specific location, is always assigned to a task manager agent on that location.

As explained in Section 7.2 the user service agent at the main location uses the yellow pages to check whether a service is available. If the service is unavailable, it is assumed that there is a communication break down. If the connection is available again, the underlying middleware will automatically publish the agents services at the remote location. The user agent receives a notification that the service is available and reconstructs the connections with the task manager agent.

### **Alternatives**

All middleware and components on a remote search location are deployed on one node. This is a potential alternative with regard to network resources in case there are only a few agents performing a search task and a few systems sharing results.

Another alternative is to have task manager agents reconstructing the connection with the user service agent after a connection break down. However, this is prohibited by the decision made Section 7.2 to use the layers pattern.

### **Assumptions**

- Line connections are not reliable and available bandwidth is not known at forehand.
- Not all signal systems are available at all locations.
- Transfer of files with task results is handled by existing file managers.

## **7.7 Mapping between views**

Figure 7.12 shows how elements used in one view were mapped to elements in other views. In addition, it shows the colours used to distinguish elements for search tasks from elements for result sharing.

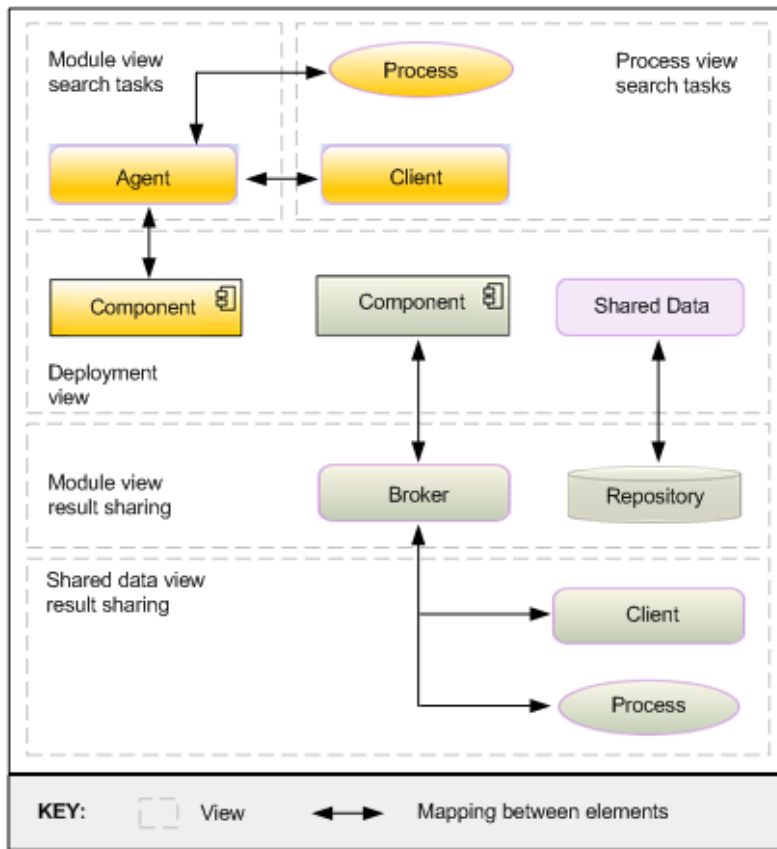


Figure 7.12: Mapping between views.

## 7.8 Conclusion

In this chapter the designed architecture is described with various views. First, the documentation across views was described. Second, the views were documented based on a standard template (SEI [9]). The selection of views was based on the most important stakeholders and their requirements.

The primary structures for the search task and result sharing system were selected in Chapter 6. The current chapter added functionality to these structures. The module, process, and shared data view showed which agent roles and repository are necessary and which functionality and interactions were needed to perform search tasks and result sharing respectively. This information, combined with the primary structures selected in Chapter 6, answers the second central question RQ 2: *Which roles and interaction models are discerned for task and result sharing?* Moreover, this chapter answers the third central question RQ 3: *Which is a proper architectural description?*

The architecture documentation of this chapter is used in chapter 8 to build two prototypes and to evaluate the architecture.

## Chapter 8

# Architecture evaluation

This chapter describes the evaluation of the architecture. Scenarios, defined in Appendix A, were used for evaluation with measuring techniques. The first section describes the prototypes that were developed for a proof of concept. The second section presents the main findings of the evaluation.

### 8.1 Prototype

We have (iteratively) developed two prototypes: *SearchOne* and *ShareOne*.

SearchOne provides search task services to users in the search process. It provides the following functionalities to the user: *(i)* add search tasks, *(ii)* show status of all search tasks, and *(iii)* remove search tasks. Signal system agents were developed for the most important signal systems. The signal systems that are used by this prototype: detection systems (M<sub>Sp</sub> and B<sub>s</sub>), recognizer system (Asset), direction finding system (Df), result sharing system (ShareOne). In addition, for each of these signal systems, signal system agents were developed that are to simulate these systems.

ShareOne provides exchange of markers in a publish/subscribe manner between entities that are decoupled in space and time. It provides the following functionalities to the user: *(i)* add filters, *(ii)* remove filters, *(iii)* show filters, and *(iv)* show (and classify) search task results. Brokers were developed for connecting to two signal systems: signal visualisation system (Tw) and search task system (SearchOne).

### 8.2 Evaluation results

Various tests with true and simulation signal systems have been performed. These tests proved all functional requirements to be realized. Only scenarios with the highest priority<sup>1</sup> have been evaluated. The results for the quality

---

<sup>1</sup>The scenarios with the highest priority, defined in Appendix A, are part of the requirements plan v1.2 12-01-2008, which is not part of this document.

## 8. ARCHITECTURE EVALUATION

---

requirements are shown in Table 8.1.

Attribute	Scenario	Result
Interoperability	Q-1.1	☑
	Q-1.2	☑
Reliability	Q-2.1	☑
	Q-2.2	☑
Efficiency	Q-3.1	☐
	Q-3.2	☐
	Q-3.3	☑
	Q-3.4	☒
Modifiability	Q-4.1	☑
	Q-4.2	☑
	Q-4.3	☐
	Q-4.4	☑
Portability	Q-5.1	☑
	Q-5.2	☑
☑	Good	
☐	Sufficient	
☒	Bad	

Table 8.1: Evaluation results of quality attributes.

### Scenarios with a sufficient score

#### *Scenario Q-4.1*

Signal system agents can detect a signal system to be idle and react properly within the maximum time. However, these agents depend on an available task manager agent (on the same location) running a search task. If this, unexpectedly, would not be the case, the signal system stays in idle mode. This could be prevented if all agents would use services of all other agents. However, this is prohibited by our implementation of the layers pattern.

#### *Scenario Q-4.3*

Task manager agents are able to react to partial results and filters within the maximum time and with the right behaviour, that is: a temporarily adjustment of the search plan preventing execution of the same subtask. However, signal systems react differently when they are rescheduled. If a signal system is performing a subtask and receives a marker to ignore

the frequency it is working on, it will continue the subtask and ignore the frequency the next time. As a result, it can not be completely prevented that two systems are working on the same frequency, despite the proper reactions of signal agents.

*Scenario Q-5.3* The addition of extra signal systems for search task execution has revealed a ripple effect. A new detector agent, which can control a new type of signal system, can be added to the system without problems. However, in order to use this new agent, the task manager agent has to be changed. The problem lies in the decision made to use a workflow as coordination mechanism to make abstract plans. A new detector agent will require a change in the task manager. Although the change is very small, it introduces a ripple effect.

### **Scenario with a bad score**

#### *Scenario Q-4.5*

The shared result system distributes events within the maximum time in a simulated environment. However, a stress test has revealed that the TuCSoN tuple space crashes if one tuple space is used by both publishers and subscribers. When data is placed in the tuple space, TuCSoN locks the tuple space and examines all coordination rules in a breadth-first approach. In the mean time, other entities write data to the tuple space buffer, that can not be processed as the tuple space is locked. As a result, the tuple space will crash. Moreover, entities can write data to the crashed tuple space without receiving an exception. The problem has been solved (for now), by splitting up the tuple space in one tuple space for publishing and one tuple space for each subscriber (See Section 7.5 in Chapter 7).

## **8.3 Conclusion**

In this chapter, the results of the architecture evaluation have been described. Two prototypes were developed for a proof of concept. Quality scenarios were used in combination with prototypes and simulation to evaluate the architecture.

Due to the fact that more signal systems were added to the prototype than initially planned, a lot of additional evaluation information was gathered. For example, the plan decomposition mechanism (workflow) required implementation changes of the task manager agent when new signal system were added.

The evaluation also revealed that the TuCSoN coordination model can be a problem in case data load increases. However, we consider this to be an implementation issue and not to be an architecture design fault. As a result, we conclude that the architecture addresses all architectural drivers.



**Part IV**  
**Discussion**

## Chapter 9

# Conclusion

In order to improve the efficiency of the search process, we have designed an architecture for a distributed system that enables both autonomous and efficient execution of search tasks and sharing of markers between several systems and users.

Maintainability and reliability were key concerns to enable an automatic and efficient search process, whereas performance was considered to be a less essential concern. The designed architecture addresses these concerns with the layers style for search task execution and the blackboard style for sharing search related information. To meet stakeholders/concerns, the model-, process-, shared data-, and deployment views were the appropriate views to describe the architecture.

The workflow-, negotiation-, and multi agent planning mechanisms were considered suitable coordination mechanisms for task sharing activities. However, the workflow mechanism for task decomposition appeared less appropriate to the issue of maintainability. The reactive coordination model was regarded as the right model to coordinate the communication between entities that share search related information with different structures. The TuCSoN middleware, used in the prototype to implement this model, reveals 'stability problems' during exposure to heavy load.

The developed (extensive) prototypes provide a lot of functionality and suitable interfaces with the most important signal systems, but lack some minor functionalities to directly deploy these prototypes in practice. However, essential parts of the prototypes can be directly deployed to obtain the desired improvement of efficiency of the search process.

## Chapter 10

# Discussion

Although we consider the selected coordination mechanisms for task sharing (workflow, negotiation, multi agent planning) suitable for use in the search process, these are not the only possible selections. Moreover, the criteria that have been defined to select these mechanisms (and models too) are far too specific for use as generic selection criteria in other domains.

We found that a reactive coordination model is very effective for sharing different kinds of (search related) information between entities. Cabri et al. [7] have used the same model. However, they used the model to exchange information with the same structure between homogeneous agents, whereas we used this model to map different information structures between heterogeneous agents. The TuCSoN coordination model we used in our prototype reveals that problems can arise during heavy load, due to a buffer overflow during execution of coordination rules. However, it can not be excluded that the coordination rules were inefficiently formulated.

Sharing partial results has shown to be an efficient mechanism for plan adjustment to prevent the execution of the same subtask. Durfee et al. [14] use the same kind of approach for result sharing between homogeneous agents that have explicit knowledge of the task structure. However, they share partial plans between homogeneous agents to integrate the partial local interpretations into a coherent overall view, whereas we shared partial results between heterogeneous agents that only have knowledge about their own task structure, in order to prevent the execution of the same subtask.

Maintainability and reliability are, in our opinion, key concerns to enable an automatic and efficient search process, whereas performance is considered to be a less essential concern. The architecture styles we selected (layers, broker, and repository) address these key concerns. Although we consider the designed architecture to fit best, it is certainly not the only architecture that can meet these concerns. For example, the technical knowledge and experience of the architect play an import role. Moreover, concerns like development costs have not been taken into account in the architectural drivers. Such considerations may influence design decisions as well.

We consider the designed system suitable for improvement of the efficiency of the search process. Although we did not prove the efficiency of the search process quantitatively, we can mention well-reasoned arguments in support of improved efficiency by this system:

1. Although (we assume that) manual search will still be performed, fewer user operations will be needed since automatic search is able to classify detected transmissions and, as a result, the user only needs to focus on the unclassified results.
2. The designed search system combines the capabilities provided by (otherwise) individual signal systems. As a result, an user can define one search task that uses multiple signal systems, instead of defining one search task for each system. In addition, the search system prevents, to a large extent, that multiple systems (that are used by that search system) are performing the same subtask.
3. The result sharing system provides sharing of search related information between systems and between systems and users. Even if this information is shared between users only, in case of manual search, it will directly improve the search process, as users can anticipate on actions of other users.
4. The search system distinguishes different types of results. Most other systems define a result when a detection and/or recognition matches the search task. However, the search system classifies a result as recognized (it matches the users search task) or not recognized (something is detected, but can not be recognized). As a result, search production will improve without user intervention.

We used a software architecture approach to systematically document the system. So far, to our best knowledge, such an approach for documenting multi agent systems has not been reported in literature before. Most agent oriented methodologies that explicitly distinguish an architecture design phase at the macro level (like Gaia [50], Mase [45], and Tropos [39]) use models that mainly produce a decomposition view. Gaia, for example, does not express characteristics like distribution, parallelism, and concurrency, as Shehory et al. [38] point out. Both Mase and Tropos use UML class diagrams, whereas Mase adds a deployment diagram to describe a multi agent system. Silva et al. [39] made a proposal to extend Tropos with component and connector views to describe the architecture, but practical experience is lacking. The usage of these agent oriented methodologies results in architectural views, aimed at using for implementation. Although these views would be sufficient for us to produce prototypes, information about why the architecture is the way it is, which alternatives were considered and why these were rejected, is lacking. As a result, the architectural documentation with these agent oriented methodologies is

---

mainly suited for developers and limited as a blueprint for construction. We used a software architecture approach in which view templates were used. This provided us with the possibility to systematically describe and substantiate the choices and decisions we made. As a result, the produced documentation provides better system understanding for various stakeholders and can, in addition, be used for system evolution. Moreover, we believe that a standard for architectural description, like IEEE 1471 [28], can complete agent oriented methodologies.

# Chapter 11

## Future work

### Future work

#### **Experimental research**

The current SearchOne prototype can perform search tasks autonomously. However, the prototype more or less depends a user that provides the search task. More intelligence can be added to agents to achieve a more autonomous search system. For example, hierarchical task networks can be used for plan decomposition instead of the workflow coordination mechanism. The task networks is a strong mechanism for uncertain situations. For example, if signal systems are or become unavailable during search task execution, task networks will try other systems to achieve its goal. Task networks can be combined with the use of goal oriented agents that use the Belief, Desires and Intention model. With regard to our designed architecture, a Jade BDI agent with task networks capabilities can be added to the task manager layer, without affecting other agents in the same or other layers. More information about task networks can be found in [15, 31]. Pokahr et al. [36] have developed Jade agents following the BDI model. Although the SearchOne prototype can be extended with more autonomy, it is unknown whether adding more intelligence to agents in the search process will actually further improve the efficiency or provide more and better results.

#### **Coordination mechanisms**

The criteria we defined to select coordination mechanisms and models were considered rather domain specific than generic. In order to develop more generic criteria to select coordination mechanisms in multi agent systems, the TEAMS coordination framework can be used to compare these mechanisms. This framework is fully described by Lesser et al. [27].



Part V

Bibliography

# Bibliography

- [1] BACHMANN, F., AND BASS, L. Introduction to the Attribute Driven Design Method. In *Proceedings of the 23rd International Conference on Software Engineering* (Washington, DC, USA, May 12 - 19 2001), IEEE Computer Society, pp. 745–746.
- [2] BACHMANN, F., BASS, L., AND KLEIN, M. Moving from Quality Attribute Requirements to Architectural Decisions. In *Proceedings of the 25th International Conference on Software Engineering (ICSE '03), 2nd International Workshop on Software Requirements to Architectures (STRAW'03)* (Washington, DC, USA, May 12-14 2003), IEEE Computer Society, pp. 122–130.
- [3] BASS, L., CLEMENTS, P., AND KAZMAN, R. *Software Architecture in Practice*, 2nd ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
- [4] BELLIFEMINE, F., POGGI, A., AND RIMASSA, G. Developing Multi-agent Systems with JADE. In *Proceedings of the 7th International Workshop on Intelligent Agents VII. Agent Theories Architectures and Languages* (London, UK, 2001), C. Castelfranchi and Y. Lesprance, Eds., Lecture Notes In Computer Science, Springer-Verlag, pp. 89–103.
- [5] BIJLSMA, A., ROUBTSOVA, E., STURMAN, S., AND JEURING, J. *Software Architecture Course*. Chapters: Software Architecture and Architectural Patterns. Open Universiteit Nederland, 2007.
- [6] BUSCHMANN, F., MEUNIER, R., ROHNERT, H., SOMMERLAD, P., AND STAL, M. *Pattern-Oriented Software Architecture: A System of Patterns*, vol. 1. John Wiley & Sons, Inc., New York, NY, USA, 1996.
- [7] CABRI, G., LEONARDI, L., AND ZAMBONELLI, F. Mobile-Agent Coordination Models for Internet Applications. *Computer* 33, 2 (2000), 82–89.
- [8] CASANOVA, H. Distributed computing research issues in Grid Computing. *SIGACT News* 33, 3 (September 2002), 50–70.
- [9] CLEMENTS, P., GARLAN, D., LITTLE, R., NORD, R., AND STAFFORD, J. Documenting Software Architectures: Views and Beyond. In *Proceedings of the 25th International Conference on Software Engineering (ICSE '03)* (Washington, DC, USA, May 03–10 2003), International Conference on Software Engineering, IEEE Computer Society, pp. 740–741.
- [10] DAVIS, R., AND SMITH, R. G. Negotiation as a metaphor for distributed problem solving. In *Distributed Artificial intelligence* (San Francisco, CA, USA, 1988), A. H. Bond and L. Gasser, Eds., Morgan Kaufmann Publishers Inc., pp. 333–356.

## BIBLIOGRAPHY

---

- [11] DENTI, E., NATALI, A., AND OMICINI, A. Programmable Coordination Media. In *Proceedings of the 2nd International Conference on Coordination Languages and Models* (London, UK, 1997), D. Garlan and D. L. Mtayer, Eds., vol. 1282 of *Lecture Notes In Computer Science*, Springer-Verlag, pp. 274–288.
- [12] DENTI, E., OMICINI, A., AND RICCI, A. Multi-paradigm Java-Prolog integration in tuProlog. *Science of Computer Programming* 57, 2 (August 2005), 217–250.
- [13] DURFEE, E. H. Distributed problem solving and planning. In *Multi-agents systems and applications* (New York, NY, USA, 2001), C. Zhang and D. Lukose, Eds., Springer-Verlag New York, Inc., pp. 118–149.
- [14] DURFEE, E. H., AND LESSER, V. R. Partial Global Planning: Coordination Framework for Distributed Hypothesis Formation. *IEEE Transactions on Systems, Man, and Cybernetics* 21, 5 (- 1991), 1167–1183.
- [15] EROL, K., HENDLER, J., AND NAU, D. S. HTN Planning: Complexity and Expressivity. In *Proceedings of the 12th National Conference on Artificial Intelligence* (Menlo Park, CA, USA, 1994), vol. 2, American Association for Artificial Intelligence, pp. 1123–1128.
- [16] EUGSTER, P. Type-based publish/subscribe: Concepts and experiences. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 29, 1 (January 2007), 6.
- [17] EUGSTER, P. T., FELBER, P. A., GUERRAOU, R., AND KERMARREC, A.-M. The many faces of publish/subscribe. *ACM Computing Surveys* 35, 2 (June 2003), 114–131.
- [18] GARCIA-OJEDA, J. C., DE J. PEREZ-ALCAZAR, J., AND ARENAS, A. E. Extending the Gaia Methodology with Agent-UML. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 1456–1457.
- [19] GELERNTER, D. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 7, 1 (1985), 80–112.
- [20] GENESERETH, M. R., AND KETCHPEL, S. P. Software agents. *Communications of the ACM* 37, 7 (July 1994), 48–53.
- [21] GONZALEZ-PALACIOS, J., AND LUCK, M. A Framework for Pattern in Gaia: A Case-Study with Organisations. In *Agent Oriented Software Engineering* (2005), J. O. et al., Ed., vol. 3382 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 174–188.
- [22] GRISS, M. L., AND POUR, G. Accelerating Development with Agent Components. *IEEE Computer* 34, 5 (May 2001), 37–43.
- [23] HEINEMAN, G. T., AND COUNCILL, W. T., Eds. *Component-based software engineering: putting the pieces together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.
- [24] JENNINGS, N. R. Coordination techniques for distributed artificial intelligence. In *Foundations of Distributed Artificial Intelligence*, G. M. O’Hare and N. R. Jennings, Eds. John Wiley & Sons, Inc., New York, NY, USA, 1996, pp. 187–210.

- 
- [25] JENNINGS, N. R., FARATIN, P., NORMAN, T. J., O'BRIEN, P., WIEGAND, M. E., VOUDOURIS, C., ALTY, J. L., MIAH, T., AND MAMDANI, E. H. ADEPT: Managing Business Processes using Intelligent Agents. In *Proceedings of the 16th Annual Conference of the British Computer Society Specialist Group on Expert Systems (ISIP Track)* (Cambridge, UK, 1996), pp. 5–23.
- [26] KRUCHTEN, P. Architectural Blueprints—The “4+1” View Model of Software Architecture. *IEEE Software* 12, 6 (Nov. 1995), 42–50.
- [27] LESSER, V., DECKER, K., WAGNER, T., CARVER, N., GARVEY, A., HORLING, B., NEIMAN, D., PODOROZHNY, R., NAGENDRAPRASAD, M., RAJA, A., VINCENT, R., XUAN, P., AND ZHANG, X. Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems* 9, 1 (July 2004), 87–143.
- [28] MAIER, M. W., EMERY, D., AND HILLIARD, R. Software Architecture: Introducing IEEE Standard 1471. *IEEE Computer* 34, 4 (2001), 107–109.
- [29] MALONE, T. W., AND CROWSTON, K. The interdisciplinary study of coordination. *ACM Comput. Surv.* 26, 1 (1994), 87–119.
- [30] MORAITIS, P., PETRAKI, E., AND SPANOUDAKIS, N. Engineering JADE agents with the Gaia methodology. In *Agent Technologies, Infrastructures, Tools, and Applications for E-Services* (2003), R. Kowalszyk, J. Miller, H. Tianfield, and R. Unland, Eds., vol. 2592 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 77–91.
- [31] MYERS, K., AND BERRY, P. At The Boundary of Workflow and AI. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99), Workshop on Agent-Based Systems in the Business context* (Orlando, Florida, July 18-22 1999).
- [32] NWANA, H. S., LEE, L. C., AND JENNINGS, N. R. Co-ordination in Multi-Agent Systems. In *Software Agents and Soft Computing: Towards Enhancing Machine Intelligence, Concepts and Applications* (London, UK, 1997), H. S. Nwana and N. Azarmi, Eds., vol. 1198, Springer-Verlag, pp. 42–58.
- [33] OMICINI, A., RICCI, A., VIROLI, M., AND RIMASSA, G. Integrating objective & subjective coordination in multi-agent systems. In *Proceedings of the 2004 ACM symposium on Applied computing* (New York, NY, USA, March 14 - 17 2004), ACM, pp. 449–455.
- [34] OMICINI, A., AND ZAMBONELLI, F. The TuCSOn Coordination Model for Mobile Information Agents. In *Proceedings of the 1st Workshop on Innovative Internet Information Systems* (1998). Pisa.
- [35] PAPADOPOULOS, G., AND ARBAB, F. Coordination Models and Languages. In *Advances in Computers* (August 1998), M. V. Zelkowitz, Ed., vol. 46, Academic Press, pp. 329–400.
- [36] POKAHR, A., BRAUBACH, L., WALCZAK, A., AND LAMERSDORF, W. Jadex - Engineering Goal-Oriented Agents. In *Developing Multi-Agent Systems with JADE* (1 2007), F. Bellifemine, G. Caire, and D. Greenwood, Eds., Wiley & Sons, pp. 254–258.
- [37] ROSSER, B. IT Architecture by Time: Today, Tomorrow or Next Minute ( Amsterdam, The Netherlands, 2000). Landelijk Architectuur Congres.
- [38] SHEHORY, O., AND STURM, A. Evaluation of modeling techniques for agent-based systems. In *Proceedings of the 5th International Conference on Autonomous Agents* (New York, NY, USA, 2001), ACM, pp. 624–631.

## BIBLIOGRAPHY

---

- [39] SILVA, C., CASTRO, J., TEDESCO, P., JO A. A., MOREIRA, A., AND MYLOPOULOS, J. Improving the architectural design of multi-agent systems: the tropos case. In *Proceedings 5th Software Engineering for Large-Scale Multi-Agent Systems (SELMAS06) in conjunction with 28th International Conference on Software Engineering (ICSE06)* (New York, NY, USA, Shanghai, China, May 22 - 23, 2006 2006), ACM, pp. 107–113.
- [40] SINGH, M. P., AND HUHS, M. N. Multiagent systems for workflow. In *International Journal of Intelligent Systems in Accounting, Finance & Management*, A. Ghose, Ed., vol. 8, nr.2. John Wiley & Sons, Ltd, 1999, pp. 105–117.
- [41] STOERMER, C., O'BRIEN, L., AND VERHOEF, C. Moving Towards Quality Attribute Driven Software Architecture Reconstruction. In *Proceedings of the 10th Working Conference on Reverse Engineering* (Washington, DC, USA, 2003), IEEE Computer Society, p. 46.
- [42] VAN AART, C. J., WIELINGA, B., AND SCHREIBER, G. Organizational building blocks for design of distributed intelligent system. *International Journal of Human-Computer Studies* 61, 5 (November 2004), 567–599.
- [43] VAN DER AALST, W. M. P., HOFSTEDE, A. H. M. T., KIEPUSZEWSKI, B., AND BARROS, A. P. Workflow Patterns. *Distributed Parallel Databases* 14, 1 (2003), 5–51.
- [44] VAN ZEIST, B., AND HENDRIKS, P. Specifying software quality with the extended ISO model. In *Software Quality Journal* (December 1996), vol. 5, nr. 4, Springer, pp. 273–284.
- [45] WOOD, M. F., AND DELOACH, S. A. An overview of the Multiagent Systems Engineering Methodology (MaSE). In *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering* (Secaucus, NJ, USA, January 2001), P. Ciancarini and M. Wooldridge, Eds., vol. 1957 of *Lecture Notes in Computer Science*, Springer-Verlag New York, Inc., pp. 207–221.
- [46] WOOLDRIDGE, M. Agent-based Computing. *Interoperable Communication Networks* 1, 1 (1998), 71–97.
- [47] WOOLDRIDGE, M., AND JENNINGS, N. R. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review* 10, 2 (1995), 115–152.
- [48] YAN, Y., MAAMAR, Z., AND SHEN, W. Integration of Workflow and Agent Technology for Business Process Management. In *Proceedings of the 6th International Conference on Computer Supported Cooperative Work in Design* (London, Ontario, Canada, July 12-14 2001), W. Shen, Z. Lin, J.-P. A. Barthès, and M. Kamel, Eds., IEEE, pp. 420–426.
- [49] YANG, H., AND ZHANG, C. Definition and Application of a Comprehensive Framework for Distributed Problem Solving. In *Proceedings of the 1st Australian Workshop on DAI: Architecture and Modelling* (London, UK, 1996), C. Zhang and D. Lukose, Eds., vol. 1087, Springer-Verlag, pp. 1–15.
- [50] ZAMBONELLI, F., JENNINGS, N. R., AND WOOLDRIDGE, M. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology* 12, 3 (July 2003), 317–370.



# Glossary

<b>AI</b>	Artificial Intelligence
<b>ASSET</b>	Asset System (for signal recognition)
<b>ATP</b>	Company specific transport protocol
<b>BS</b>	BlackStar System (for signal detection and recognition)
<b>CDPS</b>	Cooperative Distributed Problem Solving
<b>CNP</b>	Contract-Net Protocol
<b>DF</b>	Direction Finding System
<b>FIPA</b>	Foundation for Intelligent Physical Agents
<b>FIPA-ACL</b>	FIPA compliant Agent Communication Language
<b>HTML</b>	HyperText Markup Language
<b>IEEE</b>	Institute of Electrical and Electronics Engineers
<b>JADE</b>	Java Agent DEvelopment Framework
<b>MSP</b>	MScanner Pro System (for signal detection)
<b>PGP</b>	Partial Global Planning
<b>QUINT</b>	Quality in Information Technology
<b>RMI-IIOP</b>	Java Remote Method Invocation run over Internet Inter-Orb Protocol
<b>TuCSoN</b>	Tuple Centre Spread Over the Network
<b>TW</b>	TouchWood System (for signal visualisation)
<b>UML</b>	Unified Modelling Language

# List of Figures

4.1	System context . . . . .	22
6.1	Architectural pattern for task execution . . . . .	36
6.2	Architectural pattern for result sharing . . . . .	36
7.1	Module view search tasks. . . . .	42
7.2	Task assignment behaviour. . . . .	46
7.3	Context task sharing. . . . .	47
7.4	Process view search tasks . . . . .	49
7.5	Partial result behaviour. . . . .	50
7.6	Marker reaction behaviour. . . . .	51
7.7	Module view result sharing. . . . .	52
7.8	Context diagram result sharing. . . . .	54
7.9	Shared data view result sharing. . . . .	56
7.10	Publish/subscribe behaviour. . . . .	58
7.11	Deployment view. . . . .	60
7.12	Mapping between views. . . . .	63

# List of Tables

5.1	Coordination mechanisms for task sharing. . . . .	26
5.2	Coordination mechanisms against criteria . . . . .	28
5.3	Coordination models against criteria . . . . .	30
6.1	Tactic selection task execution . . . . .	34
6.2	Tactic selection for sharing markers . . . . .	34
7.1	Stakeholders relevant views . . . . .	39
7.2	Layer descriptions search tasks. . . . .	43
7.3	Agent descriptions search tasks. . . . .	44
7.4	Interface description for search task agents. . . . .	45
7.5	Connector descriptions search tasks. . . . .	50
7.6	Module descriptions result sharing. . . . .	53
7.7	Interface description for result sharing modules. . . . .	54
7.8	Connector descriptions search tasks. . . . .	57
7.9	Element descriptions deployment view. . . . .	61
8.1	Evaluation results of quality attributes. . . . .	65

Part VI  
Appendix

# Appendix A

## Quality Scenarios

In this chapter, the scenarios are described for the quality attributes specified in chapter 4.3 (Quality Attributes). Each concrete quality scenario is identified with an unique ID<sup>1</sup> and a short description of the scenario.

### Interoperability

Q-1.1 A component representing a signal system for task execution must adapt to a system specific communication protocol.

Part name	Value
Source	Developer
Stimulus	Wishes to add or change a signal system with a new communication protocol
Artefact	The component that uses this protocol to communicate with the signal system
Environment	At design time and run time
Response	No side effects
Response measure	Within four hours

Interoperability scenario for signal system search protocols.

Q-1.2 The shared memory must adapt to a system specific communication protocols.

Part name	Value
Source	Developer
Stimulus	Wishes to add or change a signal system for result sharing with a new communication protocol
Artefact	The component that uses this protocol to translate the communication between the shared memory and the signal system
Environment	At design time and run time
Response	No side effects
Response measure	Within four hours

Interoperability scenario for the shared memory.

---

<sup>1</sup>This does ID not map one-to-one to the requirements plan v1.2 12-01-2008, which is not part of this document.

## Reliability

Q-2.1 The communication between two locations is lost. The search task has to be performed in normal mode and the connection has to be reconstructed when the line is reconnected.

Part name	Value
Source	External to the system
Stimulus	Communication line between a location is unavailable
Artefact	Components
Environment	Normal operation
Response	Notify appropriate parties, continue to operate in normal mode and reconnect when the line is available again
Response measure	No down time

Reliability scenario for connection break downs.

Q-2.2 A (part of a) signal system is ceased. The parties involved are notified and the system continues performing the search task in degraded operation.

Part name	Value
Source	External to the system
Stimulus	Crash of (a part of) a signal system
Artefact	Components that are connected with the signal system
Environment	Normal or degraded operation
Response	Notify appropriate parties and continue to operate in degraded mode.
Response measure	No down time

Reliability scenario for signal system failures.

## Efficiency

Q-3.1 A component that represents a signal system detects that a signal system is idle. The component will join an existing search task within five seconds.

Part name	Value
Source	External source
Stimulus	A signal system is not performing a search task
Artefact	The component that can control the signal system
Environment	Normal mode
Response	Join an existing search task
Response measure	Within five seconds

Efficiency scenario to utilize signal systems.

Q-3.2 In case a team of components performing a search task is notified either about another team performing the same subtask or about a filter, it will adjust its search plan for the duration of the result being available or a filter being active. Action must be undertaken within five seconds.

<b>Part name</b>	<b>Value</b>
Source	External source
Stimulus	A partial result is put in the shared memory that is part of the current task plan or a component that is part of the team of one search task detects or recognizes a transmission, or an active filter
Artefact	Component that is responsible for and has knowledge about the search task
Environment	Normal mode. The component is connected with 48 detector components.
Response	Adjust search plan for the period of time that the result is available or a filter is active
Response measure	Within five seconds

Efficiency scenario for resource behaviour.

Q-3.3 A transmission is detected and a partial result is created and published.

<b>Part name</b>	<b>Value</b>
Source	Internal source
Stimulus	A transmission is detected or recognized
Artefact	Component that is responsible for and has knowledge about the search task
Environment	Normal mode. The component is connected with 48 detector components.
Response	Generate a partial result for the shared memory
Response measure	Within five seconds

Efficiency scenario for partial results.

Q-3.4 A marker is published in the tuple space. The tuple space will publish the data to all subscribers within one second.

<b>Part name</b>	<b>Value</b>
Source	External source
Stimulus	Data is placed in the tuple space
Artefact	Coordination module of the tuple space
Environment	Normal mode. Test with 20 publishers and 20 subscribers
Response	Examine coordination rules and publish data to the relevant subscribers
Response measure	Within one second

Efficiency scenario for sharing results.

## Maintainability

Q-4.1 A signal system, new to the shared memory, wants to publish or subscribe to markers. This change has to be made without rippling effects and within one day.

Part name	Value
Source	Developer
Stimulus	Wishes to add a new partial result
Artefact	Coordination module
Environment	At design time and run time
Response	Add a broker for the system and add coordination rules. No further side effects
Response measure	Within one day

Maintainability scenario for adding partial results.

Q-4.2 The structure of a marker is changed. This change must be performed without affecting other components that exchange markers through the shared memory. This has to be done within one day.

Part name	Value
Source	Developer
Stimulus	Wishes to change the structure of a marker
Artefact	Coordination module
Environment	At design time and run time
Response	Change the broker that uses this structure and change coordination rules. No further side effects
Response measure	Within one day

Maintainability scenario for changing result structures.

Q-4.3 A new signal system with specific capabilities is added to the system. A new component must be made to represent and use this system without side effects. This has to be done within three days.

Part name	Value
Source	Developer
Stimulus	Wishes to add a new signal system
Artefact	Environment
Environment	At design time and run time
Response	A new component must be added to represent the signal system without side effects
Response measure	Within three days

Maintainability scenario for adding signal systems.

Q-4.4 The way in which a signal system has to be controlled is changed. A change has to be made without a rippling effect.

<b>Part name</b>	<b>Value</b>
Source	Developer
Stimulus	Wishes to change the control of a signal system
Artefact	Environment
Environment	At design time and run time
Response	An existing component must be changed without side effects
Response measure	Within one day

Maintainability scenario for changing signal systems.

## Portability

Q-5.1 The system must adapt to Windows 2000/XP as well as Linux RH 6.x within one hour without changing the software and without losing functionality.

<b>Part name</b>	<b>Value</b>
Source	System administrator
Stimulus	Wishes to change the platform on which the system is running
Artefact	Platform (windows 2000 or XP and Linux RedHat 6.x)
Environment	At runtime
Response	Change start-up script. Keep all functionality without side effects
Response measure	Within one hour

Portability scenario for moving to another platform.

Q-5.2 More agent platforms are added at runtime in order to increase the number of components.

<b>Part name</b>	<b>Value</b>
Source	System administrator
Stimulus	Wishes to add more agent platforms to increase the number of components
Artefact	System
Environment	At runtime
Response	Change start-up script for agent platform without side effects
Response measure	Within four hours

Scalability scenario for adding platforms.

# Samenvatting

## *Achtergrond*

Het zoeken naar radiosignalen in de ether is met de huidige methoden en middelen inefficiënt gebleken. Vanwege de complexiteit van het zoekproces worden vaak identieke of overlappende taken uitgevoerd. In de praktijk blijkt dat veel signaalsystemen onvolledig benut worden. Kennis, opgedaan aan de hand van het zoekproces, wordt door het ontbreken van integratiemogelijkheden deels handmatig vastgelegd. De huidige beperkingen maken het noodzakelijk dat relatief veel manuele taken nodig zijn voor het zoeken naar signalen. Het zoekproces wordt daarom voornamelijk door gebruikers uitgevoerd. Wij willen de doelmatigheid van het zoekproces vergroten, onder andere door middel van het inzetten van autonome processen.

## *Methoden*

Voor het definiëren van criteria ten behoeve van het selecteren van coördinatiemechanismen en -modellen is gebruik gemaakt van architectuur drivers. Voor het ontwerpen van de architectuur is gebruik gemaakt van tactieken en architectuurpatronen. De architectuur is gedocumenteerd op basis van de belangrijkste belanghebbenden/eisen en geëvalueerd met scenario's en meettechnieken. Voor de ontwikkeling van prototypen is gebruik gemaakt van JADE (Java Agent DEvelopment framework) en TuCSon (Tuple Centre Spread Over the Network) middleware.

## *Resultaten*

Onderhoudbaarheid, betrouwbaarheid en prestaties zijn gedefinieerd als de belangrijkste architectuureisen. Workflow-, onderhandelings-, en multi agent planning mechanismen zijn geselecteerd om de taakdeling- activiteiten te coördineren. Een reactief coördinatie-model is geselecteerd om de communicatie tussen entiteiten, die informatie in verschillende formaten met elkaar uitwisselen, te coördineren. Teneinde te voldoen aan de kwaliteitsattributen zijn de gelaagde architectuurstijl voor uitvoering van zoektaken en de blackboard architectuurstijl voor uitwisseling van markers geselecteerd. Twee prototypen zijn, als 'Proof of Concept', ontworpen: SearchOne, welke zoektaak diensten aan gebruikers in het zoekproces faciliteert en ShareOne, welke de uitwisseling van markers, op een 'publish/subscribe' wijze, tussen entiteiten die anoniem en asynchroon communiceren, faciliteert.

### *Conclusies*

De ontworpen architectuur maakt zowel autonome als efficiënte uitvoering van zoektaken en het delen van markers tussen verscheidene systemen en gebruikers mogelijk, hetgeen resulteert in betere efficiëntie van het zoekproces. Onderhoudbaarheid en betrouwbaarheid zijn zeer belangrijke eisen om een autonoom en efficiënt zoekproces mogelijk te maken, terwijl prestaties als een minder essentiële eis wordt beschouwd. De ontworpen architectuur voldoet hoofdzakelijk aan de belangrijkste eisen.

De ontwikkelde prototypen bieden veel functionaliteit en geschikte interfaces met de belangrijkste signaalsystemen; een beperkt aantal functionaliteiten ontbreekt om deze prototypen direct in de praktijk in te kunnen zetten. De meest essentiële delen van de prototypen kunnen echter direct in de praktijk worden ingezet om de gewenste verbetering van de efficiëntie van het onderzoeksproces te realiseren.