MASTER IN COMPUTER SCIENCE

On the Use of Semantic Web Agents in Video Analysis Sharing

Hubaux, Arnaud

Award date:
2007

INSTITUT D'INFORMATIQUE
NAMUR, BELGIUM

On the Use of Semantic Web Agents in Video Analysis Sharing

Arnaud Hubaux

ANNÉE ACADÉMIQUE 2006-2007

Mémoire présenté en vue de l'obtention du grade de Maître en Informatique

ABSTRACT

Video cameras are increasingly present in our every-day environment. As a result their monitoring by human users has become basically unrealizable. At the hands of this problem, automatic processing technologies have been gradually developed and now offer reliable analysis methods. Notwithstanding, the integration and use of analysis results are currently barely tackled. Furthermore, the development of a system allowing these kinds of activities calls for particularly adapted technologies. The feasibility demonstration of such systems is consequently a daring challenge.

In order to take it up, state-of-the-art technologies are to be combined in an efficient way. Traditional object-oriented and data storage approaches may fail to offer the required flexibility and scalability. Less common and mature technologies were thus explored. The knowledge processing mechanisms we set up are based on agents. Agents are notably autonomous and reactive entities exhibiting very adaptive behaviours. The knowledge structuring and storing resulting from varying video analysis levels are based on Semantic Web technologies. The RDF graph structure appears to perfectly support our needs of *a priori* undefined metadata schema.

Moreover the integration of these components in a distributed and possibly heterogeneous environment raises concerns regarding knowledge exchange and sharing. The system we developed intends to handle these concerns by associating computer science specific concepts in order to enable real-time video analysis sharing.

KEYWORDS

agent, agent system engineering, knowledge, RDF, semantic web, video analysis

RÉSUMÉ

Les caméras videos sont de plus en plus présentes dans notre environement quotidien. Par conséquent, leur surveillance par des utilisateurs humains est devenue tout simplement irréalisable. Face à ce problème, des technologies de traitement automatique ont été progressivement mises au point et offrent maintenant des méthodes d'analyse fiables. Toutefois, l'intégration et l'utilisation de résultats d'analyse vidéo sont actuellement à peine abordées. En outre, le développement d'un système permettant ces types d'activités nécessite des technologies particulièrement adaptées. La démonstration de la faisabilité d'un tel système est par conséquent un défi audacieux.

Afin de le relever, des technologies de pointe se doivent d'être combinées de manière efficace. Les approches orientées-objets et de stockage de données traditionnelles peuvent ne pas offrir la flexibilité et l'extensibilité nécessaires. Des technologies moins utilisées et matures furent donc explorées. Les mécanismes de traitement de la connaissance que nous avons développés sont basés sur des agents. Les agents sont notamment des entités autonomes et réactives montrant des comportements très adaptifs. La structuration et le stockage de la connaissance résultant de niveaux d'analyse variables sont basés sur des technologies de Web Sémantique. La structure de graphe RDF apparaît parfaitement supporter nos besoins de non-définition *a priori* des schémas de meta-données.

De plus, l'intégration de ces composants dans un système distribué et éventuellement hétérogène soulève des préoccupations à propos de l'échange et du partage de la connaissance. Le système que nous avons développé a pour objectif de traiter ces préoccupations en associant des concepts spécifiques à l'informatique afin de permettre le partage en temps réel d'analyses vidéo.

MOTS CLÉ

agent, ingénierie des systèmes d'agents, connaissance, RDF, web sémantique, analyse vidéo

ACKNOWLEDGMENTS

To the Multitel research centre for having welcomed and followed me during five months. More particular thanks to Cyril Carincotte for his endless technical backing and help throughout my internship and the writing of this master's thesis. To Bruno Lienard for his tips and precious monitoring during development stages. To Xavier Desurmont for his constructive reviews of the carried out work.

To my promotor, Professor Pierre-Yves Schobbens, for his excellent advice and continuous support.

To Daniel Poncelet whose English skills greatly helped me to improve the quality of this document.

To my fiancée for her unending joie de vivre being a constant source of inspiration.

To my professors and mates of the Institute of Computer Science, and my family for their implicit contributions to this work.

PREFACE

In formal logic, a contradiction is the signal of defeat: but in the evolution of real knowledge, it marks the first step in progress toward victory.

Alfred North Whitehead

Video analysis and knowledge processing are both very active research areas. Recent concerns focus on the adaptation of principles from the latter to centralize and share knowledge from results of the former[1]. This work intends to propose a possible solution to these issues by means of a video analysis sharing system based on the latest knowledge management and structuring technologies. The following points will first precise the motivations underlying this project and then outline the document's global structure.

Motivations

Automatic processing of data coming from video cameras is currently a field of activity stirring up the utmost attention. State-of-the-art advances in this area enable the reliable extraction of important amounts of low-level knowledge. The analysis and interpretation of this metadata[2] allow complex event processing such as people tracking or abandoned luggage detection. Nevertheless, the way the produced knowledge is used and shared calls now for more considerations so as to bridge the gap between end-users and specific analysis algorithms. For example, a security operator may have to rapidly and dynamically analyse the produced metadata when an unexpected or important event is identified by the monitoring system without interrupting the ongoing real-time analysis. However, traditional video analysis systems are based on *a posteriori* analysis implying offline processing and preventing interactive query mechanisms.

Furthermore the lack of standards regarding low, mid and high-level analysis outputs makes tough the reuse of existing systems. They are usually targetted to specific contexts

[1] One can cite the IBM *Smart Surveillance System* (S3) [SHL+05] which illustrates them.
[2] Note that metadata and knowledge will both refer to the same elements.

and data formats and based on relational databases structured according to these predefined schemas. As a result, the exact specification of transmitted data must be *a priori* determined, ruling out any context and purpose-based data schema definition.

The main motivation of this work is to present an innovative way of handling and storing distributed video processing results in a real-time context. In order to achieve this goal, intelligent agents and data-oriented agents, named *autotroph*, were combined with the knowledge description framework used in the Semantic Web and data warehousing aspects[3]. We will demonstrate in the second part of this work how the association of these components enables the creation of a generic, context-independent and scalable knowledge sharing system.

Outline

For a long time now computer science has been struggling to find the right level of abstraction. Both the λ-calculus developed by Alonzo Church and Stephen Cole Kleene in the 30s and the Turing machine described in 1936 initiated mathematical formalisms to express algorithms. The creation and improvement of programming language paradigms ranging from machine to declarative and imperative were other breakthroughs in the evolution of software development. The growing size of computer systems leads to major advances in data, information and knowledge management, artificial intelligence and in system and requirement engineering. The repercussions they have had on mainstream computer science are commonly known.

The noticeable fact about these evolutions is the increasing abstraction degree. As systems grew and became more complex new techniques were required to keep them human understandable and manageable. Furthermore, the incontrovertible deployment of the Web offered a totally new view of data management. Data was no more this bundle of bytes resulting from the ongoing business. It has become the playground of world-scale transactions and communication. This highly interconnected net of data has caused much concern about knowledge representation and sharing.

The swift and incomplete overview of the progression of computer science presented so far directs the **first part** of this document. The four chapters it includes will present concepts of increasing granularity regarding software development.

The *first chapter* addresses the concept of agent. These entities focus the attention on many fields of activity as they enable the embodiment of human-like concepts such as goals or beliefs. Consequently much time will be spent on their definition and design. Existing languages, toolkits and the FIPA standard will also be overviewed.

The *second chapter* intends to clarify the concept of ontology and secondarily to precise the notions of data, information and knowledge, too easily mixed and misused. It will also introduce some common ontology definition languages and most notably OWL.

The *third chapter* focuses on the Semantic Web which is a hot topic of computer science as it can be the playground of many fields. Its context, definition and way of representing knowledge will be clearly investigated. Arguments vindicating the selection of the RDF will be put forward. The current status of the Semantic Web as well as its acceptance state will also be carefully discussed.

The *fourth chapter* classifies the different agent system engineering methodologies into three main categories. The most common methodologies belonging to each of them will be presented. A synthesis outlining their main characteristics and development lifecycle coverage will be performed. Finally, the current status and pitfalls of agent software engineering will be put forward.

[3]This part has been inspired by the work performed by B. Lienard in [LDBD06].

The **second part** will present a sharing system putting together the concepts described in the first part. It addresses the description of an agent-based system resorting to Semantic Web concepts and technologies to implement the sharing of video analysis results. Recent advances in video analysis now enable the relatively faithful identification of complex scenarios grabbed from video cameras. As a result, the need to centralize and benefit from the acquired knowledge becomes more and more present. The proposed solution is described in three chapters demonstrating the feasibility of the Semantic Web agent-based approach.

The *fifth chapter* thoroughly introduces the case and the chosen engineering methodology, the required concepts to its understanding, the context and the architecture of the developed system. It also describes a characteristic scenario handled by the system.

The *sixth chapter* investigates the detailed design of the system and defines the agents and events composing it. It also depicts the external and internal interactions constituting the system.

The *seventh chapter* describes in much detail the knowledge storage framework as well as the physical knowledge management system employed. The generic schema of the RDF graph as well as the specifically created autotroph agent concept will be presented.

The **third part** will synthesize both the sharing system and the theoretical parts. It is divided into two succinct chapters intending to assess the performed work.

The *eighth chapter* evaluates the solution proposed as much from the performance as from the design perspectives. Its last section will be dedicated to an overview of the possible system evolutions and extensions.

The *ninth chapter* reviews the explained concepts in order to clarify the purpose of this work.

Contents

Part I

State Of The Art

1 Agent

All our thoughts and concepts are called up by sense-experiences and have a meaning only in reference to these sense-experiences. On the other hand, however, they are products of the spontaneous activity of our minds; they are thus in no wise logical consequences of the contents of these sense-experiences. If, therefore, we wish to grasp the essence of a complex of abstract notions we must for the one part investigate the mutual relationships between the concepts and the assertions made about them; for the other, we must investigate how they are related to the experiences.

Albert Einstein

OVERVIEW

The concept of agent regularly comes over in computer science. However its definition does not seem to be unanimously acknowledged by practitioners. Before addressing this issue, a short historical overview of the agent notion will be put forward. Common types of architectures and languages used in agent-oriented applications will then be described. The last point will focus on the agent *de facto* standard, i.e. the FIPA.

1.1 Historical background

The agent fashion which seems to have boomed for a small decade relies on much older bases than we might think. Indeed, facts show that the agent perspective recurrently comes to light over time.

The first known agent was created by Joseph Weizenbaum [Wei66]. It was named *ELIZA* and was born in 1966 in the MIT labs. Its goal was to simulate a conversation between a user and a computer in the same way as a dialog with a psychiatrist. The "patient" interacts with the computer by means of a natural language, e.g. English. The main operations performed were keyword recognition, sentence transformation according to pre-defined rules and context recognition. Even if *ELIZA* was far from passing Turing's test [Tur50] it settled the premises of agents in the field of artificial intelligence (AI).

The notion of agent was also introduced in 1973 by Carl Hewitt, Peter Bishop, and Richard Steiger in their *Actor model* [Hew77]. According to their model, everything is an *Actor* evolving in a concurrent world. Each *Actor* can send messages, create new *Actors* and decide which behaviour will be applied when receiving a new message. Furthermore, the communication occurring between the *Actors* is asynchronous, which guarantees them some level of computational autonomy.

Advances in the world of AI sorely contributed to the development of agent theories. More precisely, Multi-Agent Systems (MAS) arose directly from Distributed AI (DAI), a subfield of AI. Main streams of DAI are Distributed Problem Solving (DPS), Parallel Problem Solving (PPS) and Multi-Agent Based Simulation (MABS).

Moreover, the expansion of middlewares such as CORBA or Java RMI and the development of programming languages indirectly supported the distribution of agent systems.

More recently, the use of intelligent agents in the Semantic Web, notably in the field of Web Services, has become common place as described in [CFJ+04, Hen01, MSZ01, GHS03]. [Hen99] depicts a world where human beings are represented by fully autonomous agents roaming the Internet. Even if this vision is still closer to fiction than reality, recent progress points to a massive spreading of autonomous agents on the Web. In addition, their use in bee and ant behaviours simulations [Vot04], the e-science grid [DRH04] or even in interactions between a human being and a virtual world[1] is only a few examples of their versatility.

This short overview of agent history shows that the agent technology is far from being a new paradigm. It can thus be seen as an aggregation of concepts mainly coming from AI, distributed systems and evolutions of programming languages.

1.2 Definition

Now that a brief overview of the history of agent programming has been drawn, it is time to focus on its definition. Defining of a concept used for such a long time and so widely should be easy business. Nevertheless, field specialists tend not to agree on a common definition. As presented in [FG96], we can notice a case-by-case practice in the world of agent definition.

In order to settle the meaning of this term, we will follow a two-step approach. In the first place, some definitions of agents will be introduced. In the second place, we will try to extract common meanings and lay down a definition which is general enough to meet all those previously given. Afterwards, a short introduction to agent taxonomy will be put forward.

[1] This virtual world was developed in the ALIVE project. See [Mae95] for a good introduction.

1.2.1 Various meanings

The level of accuracy of the following definitions ranges from low to high, involving wide to narrow classes of *entities* which might be considered as agents.

DEFINITION **The AIMA Agent**
An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.

Russell and Norvig defined in [RN95] their vision of software agents incarnating AI. It is easy to see that such a definition may be extremely general without any clear definitions of *environment*, *sensor* and *effectors*. Indeed any program with inputs and outputs could be assimilated to an agent.

DEFINITION **The Maes Agent**
Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed.

Maes' definition presented in [Mae95] reinforces the notions of *autonomy* and *goal*. Note that this combination is pretty straightforward as autonomy entails the need of goals to fulfill.

DEFINITION **The KidSim Agent**
An agent is a persistent software entity dedicated to a specific purpose. 'Persistent' distinguishes agents from subroutines; agents have their own ideas about how to accomplish tasks, their own agendas. 'Special purpose' distinguishes them from entire multifunction applications; agents are typically much smaller.

Smith, Cypher and Spohrer introduce in [SCS94] the concepts of *persistence* and *special purpose*. If *persistence* appears as a key point of agents, the *special purpose* feature might be too restrictive and not crucial in the definition of agents. Moreover, they tend to define agents as simple "*active objects*", which seems to be somewhat limiting.

DEFINITION **The IBM Agent**
Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires.

The definition from IBM's Intelligent Agent Strategy added the concept of *control*. This means that the behaviour of agents is guided by other entities. The *control* performed on the agent may not rule out any program from being considered as an agent, e.g. an antivirus software ordering itself to update its database every second day.

DEFINITION **The Wooldridge & Jennings Agent**
A hardware or (more usually) software-based computer system that enjoys the following properties:

- *autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;*
- *social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language;*
- *reactivity: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;*
- *pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative.*

Wooldridge and Jennings's definition was presented in [WJ95b, WJ95a]. The interesting concept introduced here is the *communication* between agents entailed by their *social ability*.

Nevertheless, some researchers, mainly from the AI field, tend to consider this definition as weak and prefer to ascribe human like concepts to agents. So besides *autonomy*, *social ability*, *reactivity* and *pro-activeness* they tack on more mentalistic notions such as *emotion*. Other attributes of agency may be:

- *benevolence*: an agent always tries to do what it is asked for;
- *mobility*: an agent can move around a network [KG99][2];
- *rationality*: an agent acts in order to achieve a goal and not to prevent it from happening;
- *veracity*: an agent does not communicate false information.

This non-exhaustive list of definitions highlights the vagueness of the agent concept. Even if they are widely used and for different purposes, the specification of a common meaning still lacks. But is it really needed ? Indeed, this fuzziness does not prevent the development of efficient applications. However, as far as system classification is concerned, clearer properties need to be settled.

1.2.2 Unified meaning

The above stated meanings show that there is no clear way to uniquely define an agent. Another approach would be to develop a taxonomy of agents as described in [FG96]. This would enable the classification of agents in groups and thus offer purpose-oriented grouping.

From the previous section arises that *acting* is a primary notion. Many entities act either in the real or virtual world, e.g. human beings, animals, some robots,... live in the real world whereas software agents live in databases, networks, OS, ... and artificial life agents live in artificial environments. What characterizes and differentiates those agents ?

Franklin and Graesser propose a list of attributes characterizing agents. As we will see, they do not define strict boundaries, which is what we expected.

[2] The interested reader may consult [CG00] from Cardelli and Gordon presenting a mobile ambient and agent calculus.

DEFINITION

Agent

An agent is an autonomous system situated in an environment that:

- *senses its environment;*
- *acts over time on its environment;*
- *is goal-driven;*
- *feels the effects of its actions on the environment.*

Figure 1.1 sketches the interactions of the agent with its environment. It can be seen that any effect on its environment will entail modifications of its behaviour through its sensors. The environment concept used since the beginning of this section calls for more accuracy. The way an agent uses its sensors and effectors on its environment restricts it sorely. What the agent senses and effects on must square with what it expects. A robot guided by sounds in a totally quiet environment can no longer be considered as an agent. Besides respecting the *agency conditions* the environment definition requires much attention. A usual way of defining environments is to use ontologies, which will be addressed in chapter 2.

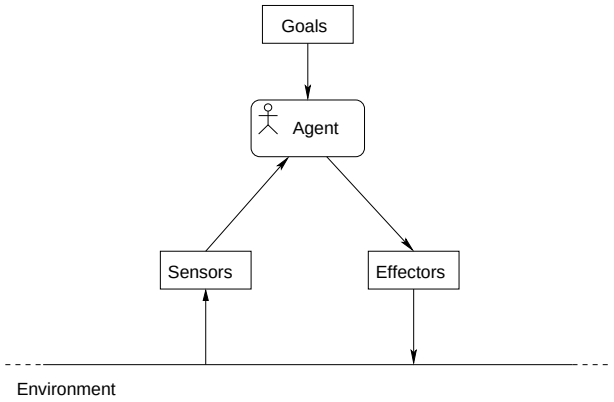


Figure 1.1 • *Agent's interactions in a scalable environment*

This agent definition includes many types of subjects ranging from human beings to thermostats. It calls consequently for more refinement. The most obvious means is taxonomy definition. Taxonomy comes from the Greek *taxis*, i.e. order and *nomos*, i.e. law or science. Such a law of order induces the notion of root or in our case of *most general agent*. It would be somewhat too restrictive again to define a single taxonomy. As we have seen, the environment as well as the system goals specify what is to be considered as an agent.

Table 1.1, adapted from [FG96], introduces some of the most common properties used to build agent taxonomies. Figure 1.2 illustrates a taxonomy presented in [FG96].

Note that objects should not be assimilated to agents. Even if objects offer a high level of abstraction, which proved to be useful in software development, they still lack many of the agent functionalities. Objects are essentially passive in nature whereas agents are active. They settle for replying to method invocations where agents consider such new information as changes in their environments. Agents are thus higher level entities which may be constituted of objects, subroutines, ... As a result, such software agents are always programs but programs must only be considered as agents if they meet the previously defined conditions.

Table 1.1 • *Common taxonomy class properties*

Property	Description
autonomous	*ability to control its actions and internal state*
character	*ability to exhibit a specific personality and emotional state*
flexible	*ability to avoid acting according to a pre-defined script*
learning	*ability to adapt its behaviour to previously encountered situations*
mobile	*ability to move over a network*
persistent	*ability to run over time without being told to*
pro-active	*ability to achieve its goal without necessarily being told to*
reactive	*ability to respond in a timely fashion to environment stimuli*
social	*ability to communicate with other agents*

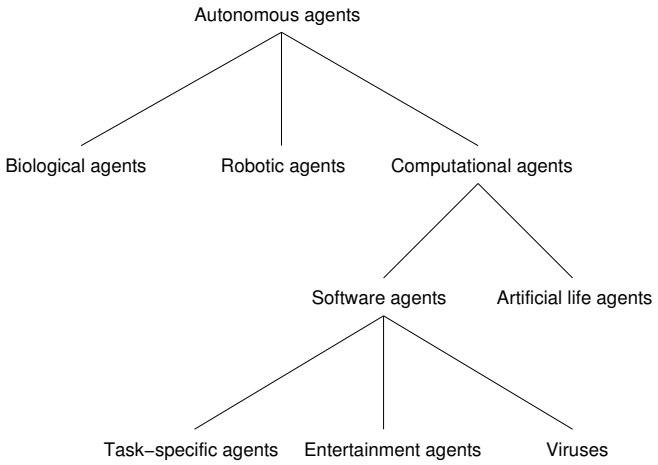


Figure 1.2 • *Example of agent taxonomy*

A more formal specification of agents is out of the scope of this document but the interested reader will find a detailed introduction to this topic in [WJ95b].

1.3 Architecture

Definitions, even the most accurate ones, are of no use if there is no way to apply them. The goal of this section is to turn those theories into working elements. These working elements will be specified by means of architectures [LAD04]:

DEFINITION

Agent architecture
Architectures provide information about essential data structures, relationships between these data structures, the processes or functions that operate on these data structures, and the operation or execution cycle of an agent

Firstly, the emphasis will be put on the agent's micro architecture, i.e. its internal

structure. There are three commonly accepted types of architectures: *deliberative*, *reactive* and *hybrid*. Secondly, more attention will be paid to the macro architecture, i.e. the way agents interact with each other. Given that we do not intend to exhaustively list the available ones, the focus will be put on the most well known and used architectures.

1.3.1 Deliberative architecture

This architecture can be defined as follows [WJ95b]:

DEFINITION

Deliberative agent architecture

One that contains an explicitly represented, symbolic model of the world, and in which decisions (for example about which action to perform) are made via logical (or at least pseudo-logical) reasoning, based on pattern matching and symbolic manipulation.

In other words, some mentalistic aspects are assigned to agents. These attitudes are grouped into three categories [LAD04]:

- *informative*: what is considered to be true about the world, i.e. assumption, belief and knowledge;
- *motivational*: what is wanted, i.e. desires, goals and motivations;
- *deliberative*: what directs behaviour, i.e. intentions and plans.

This category of agents gave birth to the *Belief Desire Intention* (BDI) model. BDI agents possess an evolutive set of *beliefs* about the current world. In addition to these beliefs, a set of *desires* has to be fulfilled. However, given current beliefs some desires can not be achieved. Agents have thus to determine a subset of desires that might be met, i.e. the *intentions*.

Most of the successful agent systems' architectures are based on the BDI model. The *Intelligent Resource-Bounded Machine Architecture* (IRMA) project [BIP91] and the *Procedural Reasoning System* (PRS) initially presented in [GL87] are evidential examples. Figure 1.3 illustrates the PRS architecture where goals are desires and system operationalization is performed by means of plans. So plans are selecting desires to become intentions according to the state of the belief database.

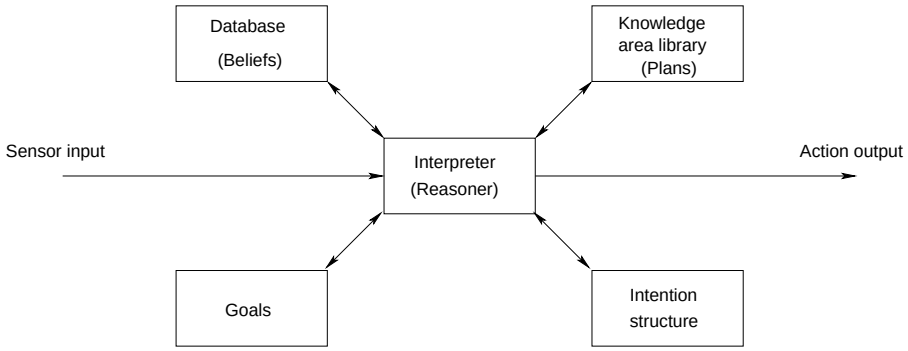


Figure 1.3 • *The Procedural Reasoning System architecture*

Further investigations about BDI based architectures may be found in [WJ95b, LAD04].

1.3.2 Reactive architecture

As stated in [WJ95b]:

DEFINITION **Reactive agent architecture**
One that does not include any kind of central symbolic world model, and does not use complex symbolic reasoning.

The AI view presented in the previous point induces a symbolic representation of the world to exhibit intelligent behaviours. To determine the best action to perform much symbolic reasoning is needed, which is very resource and time consuming. As a result, the action chosen may no longer be the most appropriate one given the modifications of the environment since the beginning of the reasoning. Reactive architectures are just the opposite, it is only when placed in the real environment that the system is able to respond to stimuli and that effective behaviours can occur.

Unlike deliberative architectures where reactive agents' behaviour is determined at run time, reactive architectures define behaviours at construction time.

Brooks was one of most fervent opponents of the deliberative model. He described in [Bro85] the *subsumption architecture* which is presented in figure 1.4. His architecture establishes a hierarchy of *task-achieving behaviours* competing with each other to control the agent (a robot in the case of figure 1.4) where tasks from higher levels are more specific than those from lower levels.

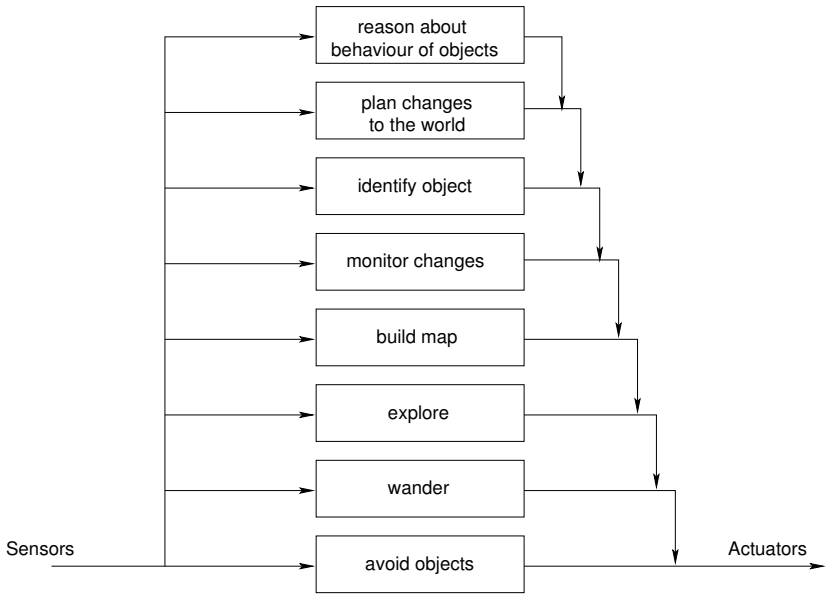


Figure 1.4 • *The subsumption architecture*

Rosenschein and Kaelbing [RK86] followed another approach which is the *situated automata* where agents are defined in declarative terms. Once specified, those declarations are compiled into a *digital machine* operating in a *time-bounded fashion*.

However, the scalability of such *ad hoc* systems may be questioned if more complex solving behaviours are required. Other types of reactive architectures are discussed in [LAD04, WJ95b].

1.3.3 Hybrid architecture

Given that reactive architectures may be inadequate to model sophisticated behaviours and that deliberative architecture may not be able to respond timely fashion, the emergence of an alternative architecture was fairly natural:

DEFINITION

Hybrid agent architecture
One that benefits from both:

- *the timely fashion reactiveness in changing environments of reactive architectures;*
- *the best progression of action selection in stable environments of deliberative architectures.*

A key application of such architecture is the *TouringMachines* developed by Ferguson in his PhD thesis [Fer92] and depicted in figure 1.5.

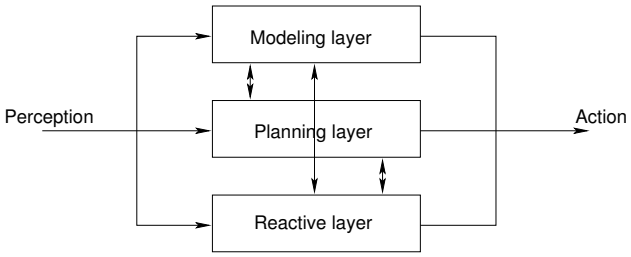


Figure 1.5 • *The TouringMachines architecture*

This three-layer architecture offers all the reactivity of reactive architecture through the *Reactive layer*. The *Planning* and *Modeling layers* enable the handling of plans and the maintenance of the environment model.

It is argued in [WJ95b] that the *ad hoc* construction and the competition of the various layers of such constructs prevent the establishment of formal theories.

Other examples of hybrid architectures such as *InteRRaP* are discussed in [LAD04, WJ95b].

1.3.4 Distributed architecture

So far we have only considered agents' internal architectures leaving aside real communication among agents. Distributed architecture aims at presenting a more *holistic* approach of agent system design. If we adapt the concept presented in [LAD04]:

DEFINITION

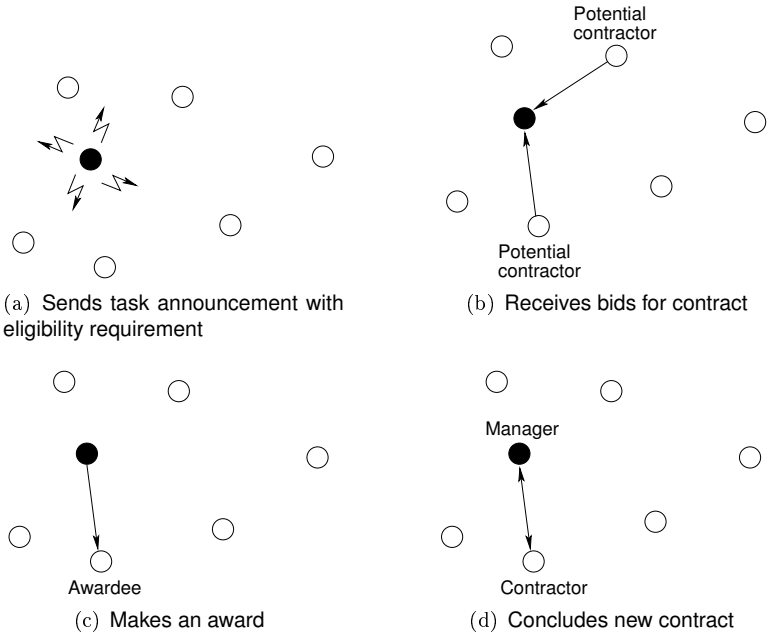
Distributed agent architecture
One that focuses on MAS where the global system structure is the main concern. It emphasizes the interaction, communication and coordination between agents.

One of the most well known coordination mechanism is probably the *Contract Net Protocol* (CNP) [Smi80]. The CNP was designed to handle communication and coordination between distributed entities such as agents[3] and aims at solving the *connection problem* which is:

[3] Originally, the CNP was *developed to specify problem-solving communication and control for nodes in a distributed problem solver.*

DEFINITION **Connection problem**
Discovery of a means whereby nodes with tasks to be executed can find the most appropriate idle nodes to execute those tasks.

The goal of the CNP is thus to dynamically allocate tasks to the most suitable agents in the network by establishing contracts between them. Figures from 1.6(a) to 1.6(d) describe the basic steps of the CNP. Other approaches such as *Agentis* are discussed in [LAD04].



(a) Sends task announcement with eligibility requirement
(b) Receives bids for contract
(c) Makes an award
(d) Concludes new contract

Figure 1.6 • *The Contract Net Protocol*

1.4 Languages

This section will focus on *Agent-Oriented Programming Languages* (AOPL). In such techniques, it is the language that builds in mental attitudes, such as transition rules, which determine how the agent will react to some given inputs. As a result, its mental state is continuously updated and outputs adapted accordingly. AOPL[4] could be defined according to [MRM02, WJ95b, LAD04] as:

DEFINITION **Agent-Oriented Programming Language**
Language facilitating the development of agent system by implementing agent related concepts such as belief, goals, intention or other mentalistic notions. They thus offer a higher level handling of agent properties.

The aim of the following subsections is to pinpoint some of the AOPL which defined the basis of modern languages. The interested reader should explore further through the references and consult [WJ95b, LAD04] which investigate this subject more thoroughly.

Note that more recent and actually used languages will be addressed in the toolkits part.

[4] Note that this definition suggests the use of a deliberative architecture.

1.4.1 Concurrent Object Languages

As stated in point 1.1, agents greatly benefited from the evolution of programming languages. Object-oriented (OO) languages can be regarded as ancestors of AOPL. Indeed they enable the construction of self-contained entities executing concurrently, the protection of internal states and the response to messages coming from entities featuring the same properties. These notions are closely linked to what constitutes current agents. The Hewitt's *Actor Model* [Hew77] was one of the first frameworks implementing an OO language.

1.4.2 Agent0

Shoham defined in [Sho93] a new paradigm promoting the social view of the system where he emphasizes the fact that the agent is constituted of *mental components*. The language he developed from his theory is *Agent0* where the agent is specified in terms of capabilities (what the agent can do), initial beliefs and commitments, and commitment rules (how the agent acts).

1.4.3 PLACA

The *Planning Communicating Agents* (PLACA) language was developed in Thomas's PhD Thesis in 1993 [Tho93]. She intended to extend *Agent0* to avoid its inability *to plan, and communicate requests for actions via high level goals* [WJ95b]. To fulfill this, operators were introduced for action planning and goal achievement. PLACA agents are defined by mental states and mental changing rules.

1.4.4 Concurrent MetateM

Concurrent MetateM was developed by Fisher [Fis94]. Unlike the two previous languages, agents are here specified in a temporal logic which is directly executable. The specification is thus directly executed to generate the agent's behaviour. Each agent is defined as a concurrently running process communicating with other agents by means of message-passing. As agents evolve in a dynamic environment, changes are incorporated into the system according to an *execution strategy*.

1.4.5 Telescript

Telescript was maybe the first commercial agent language [WJ95b]. It was developed by General Magic in the early 90s which was an Apple Computer spin-off. The two key concepts of *Telescript* were *agents* and *places*. Agents are seen as communicating mobile processes moving from one place to another. They both provide and consume goods in *electronic marketplace* applications Telescript was supporting. It is backed by a language, an interpreter, a protocol set and development tools.

1.4.6 Toolkits

Agents toolkits, more than specifying an AOPL, provide sets of tools and developing environments that help the creation of complex systems. Some of the most common ones are: IMPACT [IMP05], JADE [BCT+06], JACK [Age06], RESTINA [Syc02], Zeus [CNvB00] ... [LAD04] details each of them in a very wise manner.

1.5 Standards

The *Foundation for Intelligent Physical Agents* (FIPA) started in 1996 and has been an IEEE Computer Society standards organization since March 2005.

The FIPA intends to specify standards for agent and MAS and is considered as the *de facto* standard in this field. The areas covered by the FIPA are presented in Figure 1.7 and are explained in one or more specification documents [FIP05].

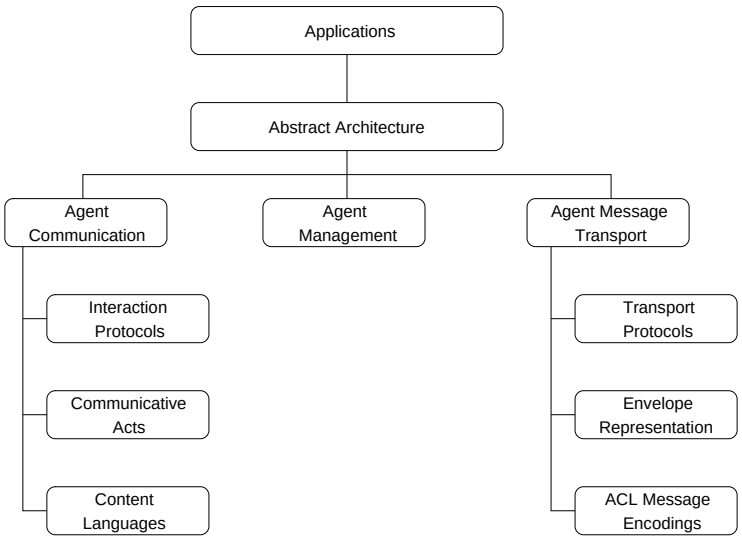


Figure 1.7 • *Overview of the FIPA specification taxonomy*

The *Application* specification documents addresses, notably in `FIPA00014`, `FIPA00091` and `FIPA00094`, the way agents should be developed according to the application type.

The *Abstract Architecture* (`FIPA00001`) aims at presenting an overall description of the FIPA standard set and thus *to foster interoperability and reusability*. It is divided into three parts:

1. *Agent Management*: `FIPA00023` *provides the normative framework within which FIPA agents exist and operate* and consists of an agent run-time environment, an agent management system, a directory facilitator and a message transport system.

2. *Agent Message Transport*: is divided into several standards such as `FIPA00067`, `FIPA-00069`, `FIPA00070`, `FIPA00071`, `FIPA00075`, `FIPA00084` and `FIPA00088` which focus on the low-level message transport.

3. *Agent Communication*: focuses on the meaning of the messages passed by the transport layer and is mainly defined in `FIPA00008`, `FIPA00037` and `FIPA00061`.

2 Ontology

How could drops of water know themselves to be a river? Yet the river flows on.

Antoine de Saint-Exupéry

OVERVIEW

Information systems, like agent systems, are based on knowledge or less restrictively on data handling. Arguably data structure definition is the keystone of an efficient system matching specific requirements. Furthermore, one has to admit that concept naming is far from respecting pre-defined rules. As a matter of fact, it is usual to encounter one term defining several concepts or different terms defining the same concept across various systems. Such issue is also known as the *Tower of Babel problem* [Smi03a]. This variety of meanings associated to knowledge sharing expansions entails the needs of more unicity and consistency. In other words, it implies the emergence of common *ontologies*, which will be the object of the first section. The second section will focus on ontology creation languages and more precisely on OWL.

2.1 Definitions

Firstly, this section will attempt to provide a definition of ontology. Secondly, the concepts of data, information and knowledge will be addressed. Their careless use in computer science leads to confusions, which requires some clarifications. However, the definitions proposed simply aim at making their meanings clearer and do not intend to thoroughly investigate the subject.

2.1.1 Ontology

Like many terms used in computer science, the term *ontology* was borrowed from another domain. The word ontology was coined in 1613 by two separate philosophers: Rudolf Göckel in his *Lexicon philosophicum* and Jacob Lorhard in his *Theatrum philosophicum*. Ontology comes from the Greek *ontos*, i.e. to be and *logos*, i.e. science. Ontology is often assimilated to *metaphysics*[1] and is thus seen as the science of *being* or *reality*. Note that *metaphysics* does not have to be confused with *epistemology* which studies the nature of knowledge.

According to Barry Smith [Smi03a] the definition of ontology is:

DEFINITION **Ontology (Philosophy)**
The science of what is, of the kinds and structures of objects, properties, events, processes and relations in every area of reality.

Thomas Gruber defined an ontology as [Gru93]:

DEFINITION **Ontology (Computer science)**
An explicit specification of a conceptualization.

where conceptualization has to be seen as (adapted from [Gru93]):

DEFINITION **Conceptualization**
Definition of an abstract and simplified view of the world to model. This view includes entities, e.g. objects or concepts existing in the world and the relationships among them.

Ontologies are thus used to specify conceptual entities described by data rather than to specify data.

Gruber also delineates five ontology design criteria:

1. *clarity*: in order to effectively specify the defined terms, the definition should be objective, complete and respect some kind of formalism;
2. *coherence*: the inferences performed will not contradict with the ontology definition. Furthermore, there cannot be inconsistencies between informal and formal definitions;
3. *extendability*: the definition should be structured in a way allowing further evolutions over time;
4. *minimal encoding bias*: the language used to express the definition should not influence its definition. In other words, bias due to a particular encoding must be ruled out;

[1] Literally meaning: *what comes after the physics* which was called by Aristotle *the first philosophy*.

5. *minimal ontological commitment*: the definition should do as few claims as possible about the world to avoid too restrictive applications. It must be weak enough to tolerate various instantiations and precise enough to exclude inappropriate models.

It may appear manifest that meeting all those criteria in a single ontology is utopian and that *tradeoffs* are mandatory. However, those seeming contradictions may not be as obvious as what could be expected. For example, on the one hand *clarity* entails the detailed and accurate definition of terms which might imply strong restrictions. On the other hand, *ontological commitment* entails the definition of a weak theory which might imply many possible models. Attention should be paid to the fact that *clarity* focuses on terms whereas *ontological commitment* focuses on conceptualization. As a result, a weak model definition does not rule out strongly defined concepts.

Ontology case studies developed in KIF are presented in [Gru93]. They are followed by discussions about the chosen design methods. More advanced consideration on the evolution of ontology in the world of Information Science may be found in [Smi03a].

2.1.2 Related concepts

As the definition of ontology has been established, some terms regularly coming over throughout this document, and more generally in computer science, have to be precised. T.S. Eliot wrote in an opening stanza from choruses in [Eli34]:

> ...
> *Where is the wisdom we have lost in knowledge?*
> *Where is the knowledge we have lost in information?*
> ...

which might be extended with:

> *Where is the information we have lost in data?*

A conceivable interpration of this quotation is that the focus we put on knowledge drives us to forget about the wisdom required to manage it. This reasoning can be extended until we reach data, i.e. the smallest unit handled by systems. Arguably this view matches the one supported by previous computer systems only focusing on raw data structuring and storage. Nevertheless, current trends in *Business Intelligence* (BI) tend to promote information and more recently knowledge as the fundamental concern in software design. As the concepts of *data*, *information* and *knowledge* depend on different fields of activity they must be clearly identified. The approach followed so far could be summarized by the following *transformation chain*:

$$data \rightarrow information \rightarrow knowledge$$

Note that the presented definitions may appear somewhat limiting. Their intent is only to clarify borders between these terms and make their conceptual links precise. The interested reader should consult [Hey04] and [DP00], from which the following definitions were adapted, for a more complete introduction to the topic.

The first link in this chain is with no contest *data* that can be defined as:

DEFINITION

Data

Is a set of discrete and objective facts about events that are most of the time classified and stored. Data offers possibilities of quantitative and qualitative assessments. Nevertheless, it provides no judgement, no interpretation and no sustainable basis of action.

Therefrom comes out that data only represents facts and says nothing about its importance or relevance. However, it is the essential raw material used to build information. Information comes from the Latin *informare* which means to give shape or more precisely:

DEFINITION **Information**
Is a message that may have a written, audio or video form originating from a sender and dedicated to a receiver. To be valuable the message must have an impact on the receiver. It is the value added to data by means of contextualisation, categorization, calculation, correction and condensation that builds up meaning and thus creates information. Although the medium affects the message, it does not have to be considered as a part of the message.

For information to become knowledge, a human intervention is required. More precisely, it is human beings who convert information into knowledge which might be defined as:

DEFINITION **Knowledge**
Is a mix of framed experience, values, contextual information and expert insight that provides a framework for evaluating and incorporating new experiences and information. It originates and is applied in the mind of knowers. It may be embedded in documents, repositories, organizational routines, processes, practices or norms. Knowledge exists within people, part and parcel of human complexity and unpredictability. The transformation from information to knowledge is achieved through comparison, consequences, connections and conversation analysis.

One might argue, rightly, that knowledge is not only the privilege of human beings. Unfortunately, this document is not the most appropriate place to lead such a debate. As we will see further, the term *knowledge* will be used in a much more technical context. Researchers in cybernetics and AI intending to simulate the human behaviour unavoidably included human related notions into their terminologies. Consequently, the knowledge concept extended somewhat naturally to the whole computer science domain.

2.2 Languages

This section will briefly introduce some of the most common ontology definition languages. OWL, which is becoming the *lingua franca* of the Semantic Web, will be investigated more thoroughly.

2.2.1 CycL

Cyc corporation aims at *constructing a foundation of basic "common sense" knowledge–a semantic substratum of terms, rules, and relations–that will enable a variety of knowledge-intensive products and services* [CYC]. The ontology definition language *CycL* is specified in [Ope] and intends to allow applications to perform human-like reasonings. The Cyc project has been widely criticized notably for its complexity due to its encyclopedic ambition.

2.2.2 DAML+OIL

The *DARPA Agent Markup Language* (DAML) project began in 2000 and ended in early 2006. The goal of DAML was to develop tools and a language supporting the development of ontologies. DAML extends XML [XML] and RDF[2] [RDF04] to enable the creation of

machine readable ontologies: *DAML provides a basic infrastructure that allows a machine to make the same sorts of simple inferences that human beings do* [DAMa]. It was also strongly integrated in the Semantic Web and mainly in Web service descriptions.

Ontology Inference Layer (OIL) proposes four ontology languages with increasing completeness levels [OIL, FvHH+01]. It intends to combine modelling primitives from frame-based languages (FBL) and reasoning services provided by description logics, which will be defined in point 2.2.6. FBL can be defined as (adapted from [WFCF88]):

DEFINITION **Frame-Based Language**
A frame can be used to represent AI structures, objects and classes but it differs from, for instance, a property list in that one cannot (usually) add arbitrary data to a frame. Moreover, not all slots in it need to be filled. It also has some facilities for inheritance and attached procedures definitions and is commonly context aware. FBL enable the syntactic and semantic enactment of these concepts.

A characteristic example of FBL is an OO language like Java.

DAML+OIL is the combination of DAML and OIL and is defined as *a semantic markup language for Web resources* [DAMb, DAMc, HPSvH02]. Note that it is now superseded by OWL[3].

2.2.3 KIF

The *Knowledge Interchange Format* (KIF) is *a language designed for use in the interchange of knowledge among disparate computer systems (created by different programmers, at different times, in different languages, and so forth)* [Stab]. It does neither intend to support interactions with users nor to be used as internal structure of knowledge within computer systems. In other words, KIF structures have to be converted into internal structures such as arrays or lists before being handled.

2.2.4 Ontolingua

Ontolingua *provides a distributed collaborative environment to browse, create, edit, modify, and use ontologies* [Ont]. More information about Ontolingua ontology creation and handling is available in [Staa]. Note that it is backed by a KIF parser.

2.2.5 SHOE

As defined in [HHL99] *Simple HTML Ontology Extensions* (SHOE) is a knowledge representation (KR) language allowing the introduction of semantic annotations in Web pages. SHOE aims at solving inherent web page problems which are: *lack of structure*, *heterogeneity* and *contextual dependency*. It also intends to make Web pages understandable and automatically processable by specialized intelligent agents.

2.2.6 OWL

The *Web Ontology Language* (OWL) is defined according to [OWL04a] as:

[2] For the moment, RDF will simply be considered as a graph-based data structure. A more relevant definition will be proposed in point 3.3.

[3] An overview of changes from DAML+OIL to OWL is presented in Appendix D of [OWL04b] and its limitations are discussed in [Smi03b].

DEFINITION **OWL**

Is an ontology definition language that:

- *is intended to be used when the information contained in documents needs to be processed by applications, as opposed to situations where the content only needs to be presented to humans;*
- *can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms;*
- *has more facilities for expressing meaning and semantics than XML, RDF, and RDFS, and . . . goes beyond these languages in its ability to represent machine interpretable content on the Web.*

OWL was created by the *Web Ontology (WebONT) Working Group* which was chaired by James Hendler. Its development started in 2001 and the latest recommendation dates from February 2004. Unlike previous languages, OWL is a Web-dedicated ontology language. Combined with RDF, OWL adds the following properties to ontologies [Hen04]:

- *ability to be distributed across many systems*;
- *scalable to Web needs*;
- *compatible with Web standards for accessibility and internationalization*;
- *open and extensible.*

OWL offers three *increasingly expressive sublanguages*: OWL Lite, OWL DL and OWL Full which will be developed hereafter. Figure 2.1 illustrates how the different OWL layers overlap. Note that every layer can express the same things as its sublayers and can draw the same conclusions.

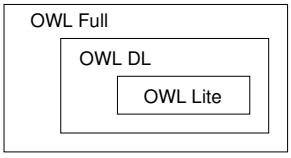


Figure 2.1 • *OWL expressiveness layers*

The last point will outline the evolution of OWL, i.e. OWL 1.1.

OWL Lite

OWL Lite only supports a restricted subset of OWL and is mainly targeted at developers needing simple constructs or classification hierarchy. It privileges lower expressivity and higher efficiency which is required to build complete reasoners[4].

Table 2.1 contains an overview of the syntactic constructs available in OWL Lite.

[4] An example of context reasoning based on OWL may be found in [WZGP04].

OWL DL

OWL DL holds its name from *Description Logic* (DL). DL can be defined according to [Lam] as:

DEFINITION **DL**

Description logics are knowledge representation languages tailored for expressing knowledge about concepts and concept hierarchies. ... The basic building blocks are concepts, roles and individuals. ... Description logic systems have been used for building a variety of applications including conceptual modeling, information integration, query mechanisms, view maintenance, software management systems, planning systems, configuration systems, and natural language understanding.

OWL DL, and consequently OWL Lite, foundations on DL guarantee *computational completeness*[5] and *decidability*[6]. Advanced considerations on OWL Lite/DL and DL are developed in [HPS04b].

Table 2.2 contains an overview of the additional syntactic constructs available in OWL DL. Unlike OWL Full, some restrictions on the use of those constructs have been set and outlined in [OWL04b]. [HPS04a] presents also some of OWL's limitations and extends it to form the *OWL Rules Language* intending to make up for them.

OWL Full

OWL Full releases all restrictions active in OWL Lite/DL. As a result, no *computational* guarantee can be assured.

Table 2.2 contains an overview of the additional syntactic constructs available in OWL Full.

OWL 1.1

The new draft version of OWL, OWL 1.1, which is maintained at [GBC+] has been set up in order to meet users' claims about OWL limitations. More precisely, OWL 1.1 is intended to be the extension of OWL DL.

The four main categories of features added to OWL DL are [PSG06]:

1. *Syntactic sugar*: new constructs, `DisjointUnion` and `valueNot`, ease the writing of some common idioms;
2. *New description logic construct*: recent advances in DL increase its expressive power by adding properties such as reflexivity/irreflexivity;
3. *Datatype expressiveness*: new constraints can be added on datatypes;
4. *Meta-modelling constructs and annotations*: restrictions on individuals, class and property names are released. They can now all have the same name without losing computional guarantees.

[5] Computational completeness entails that all conclusions are assured to be computed.
[6] Decidability entails that all computations will be performed in finite time.

Table 2.1 • *OWL Lite language constructs*

(a)

RDF Schema Features
Class (Thing, Nothing)
rdfs:subClassOf
rdf:Property
rdfs:subPropertyOf
rdfs:domain
rdfs:range
Individual

(b)

(In)Equality
equivalentClass
equivalentProperty
sameAs
differentFrom
AllDifferent
distinctMembers

(c)

Property Characteristics
ObjectProperty
DatatypeProperty
inverseOf
TransitiveProperty
SymmetricProperty
FunctionalProperty
InverseFunctionalProperty

(d)

Property Restrictions
Restriction
onProperty
allValuesFrom
someValuesFrom
equivalentClass
equivalentProperty
sameAs
differentFrom
AllDifferent
distinctMembers

(e)

Restricted Cardinality
minCardinality (only 0 or 1)
maxCardinality (only 0 or 1)
cardinality (only 0 or 1)

(f)

Header Information
Ontology
imports

(g)

Versioning
versionInfo
priorVersion
backwardCompatibleWith
incompatibleWith
DeprecatedClass
DeprecatedProperty

(h)

Annotation Properties
rdfs:label
rdfs:comment
rdfs:seeAlso
rdfs:isDefinedBy
AnnotationProperty
OntologyProperty

(i)

Class Intersection
intersectionOf

(j)

Datatypes
xsd datatypes

Table 2.2 • *OWL DL/Full additional language constructs*

(a)

Class Axioms
oneOf (dataRange)
disjointWith
equivalentClass (applied to class expressions)
rdfs:subClassOf (applied to class expressions)

(b)

Boolean Combinations of Class Expressions
unionOf
complementOf
intersectionOf

(c)

Arbitrary Cardinality
minCardinality
maxCardinality
cardinality

(d)

Filler Information
hasValue

3 Semantic Web

If we value independence, if we are disturbed by the growing conformity of knowledge, of values, of attitudes, which our present system induces, then we may wish to set up conditions of learning which make for uniqueness, for self-direction, and for self-initiated learning.

Carl Rogers

OVERVIEW

From previous chapters arises that agents and ontologies are highly related concepts. As a result, many applications resort to those technologies to develop complex and distributed systems. The most patent example of this combination is probably the *Semantic Web* (SW). Firstly, we will contextualize and define the concept of SW. Secondly, we will present the data format employed in this world-scale project and vindicate its selection. Finally, the current status of the SW will be assessed.

3.1 Contextualisation

The consciousness of the context on which the SW is based is a key step toward its proper understanding. As a matter of fact, its definition would not be relevant if not preceded by an overview of the main Web evolutions which leaded to its conceptualization.

Let us try to find out how the Web is "perceived". A common trip on the Internet implies much clicking, various context browsing, information sorting and understanding. The most natural means to achieve such operations is by using a language you easily master. Human beings are rather "developed " beings that have, among other things, the opportunity to view and hear contents. Such capabilities allow one to process the natural language(s) in which Web pages are written, watch videos or listen to music, i.e. to process multimedia material. Processing means in this case combining, deducing and creating associations between facts and (partial) information. These are easy tasks for a human being[1].

From those observations it can be noticed that the role of technologies is fairly static and dedicated to information display. For example, how could a search engine automatically combine information, perform analogies between concepts or include multimedia content into its search ? If we consider a lower level of abstraction, how could it know that <auto:buyer> is almost equivalent to <car:owner>[2] ? As a consequence, search results may be far from what you could have expected just because the most widely used terminology is not the one you know, is rarely employed or is simply massively used to market some products. Another known issue is multimedia finding. Indeed, the only matchings performed are based on file names and not on contents, and one knows that names may not be relevant to the explicit content[3].

In addition to those search issues, form filling is another common pervasive problem. Effectively, if field contents are nearly always the same, their labels may vary. There are numerous times when such information supply is required but the automatic storage and filling is prevented by this non-uniformity of meanings or at least tagging.

Less customer oriented considerations should also be addressed. Indeed, business-to-business (B2B) applications also suffer from the lack of uniformity. Even if protocols such as *Simple Object Access Protocol* (SOAP) [SOA03] are developed to allow the invocation of Web services in a standardized way, they do not solve the problem of service description consistency. As well as Web content, Web service description and sharing calls for more coherence and formal description.

From the above-mentioned observations comes out that mechanisms should be deployed to create self-describing data. What intuitively comes to mind is that ontologies could be a means to achieve this goal. Indeed, *ontologies are a means to make an explicit commitment to shared meaning among an interested community* [SHBL06]. Moreover, the precise definition of interrelated concepts would be of no use if not coupled with efficient reasoning systems. Such formal reasoning entails the specification of a well-defined and storage-independent data structure, which will be the topic of point 3.3. Furthermore, this search evolution could involve many semantically-linked pages. All this reasoning must be handled by dedicated and specialized entities such as agents.

[1] Attention should be paid to the fact that some disabilities prevent such uses of the Web. [W3C05] presents some of them and introduces alternative means to them to browse the Internet.

[2] If we consider here the buyer of the car as its owner.

[3] [NRST+06] depicts methods for multimedia content description by means of ontologies while [SDWW01] focuses on picture annotation.

3.2 Definition

Probably the first article, called *The Semantic Web* [BLHL01], presenting the SW was published in the *Scientific American* in 2001. It lays down the foundations of the SW in the continuity of the efforts made on the Web[4].

At this point, it is important to understand that the SW is not intending to develop some kind of parallel Web. On the contrary, it aims at giving the current Web a more structured and consistent definition. As presented in [HBLM02], the SW can be seen as:

DEFINITION

Semantic Web

Is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation. It is based on the idea of having data on the Web defined and linked such that it can be used for more effective discovery, automation, integration, and reuse across various applications. . . . [It] will provide an infrastructure that enables not just web pages, but databases, services, programs, sensors, personal devices, and even household appliances to both consume and produce data on the web.

Figure 3.1 sketches the SW structure originally presented by Tim Berners-Lee in [BL00] at the *XML 2000 Conference*.

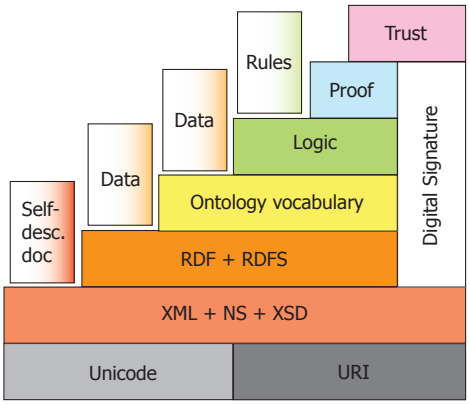


Figure 3.1 • *The Semantic Web layered structure*

If we ride up this stack, the following layers successively appear:

- *Unicode/URI*: basics of the Web as they define data encoding and hyperlink formats;
- *XML+NS+XSD*: data format specifications enabling the creation of advanced data structures by providing efficient means to present information;
- *RDF+RDFS*: data format specifications constituting the basics of the SW as they are used to structure the shared knowledge;
- *Ontology vocabulary*: meta-information enabling the complex definition of interoperable ontologies;

[4] The interested reader should consult [BLM02] that highlights some projects which contribute to the development of the SW.

- *Logic*: inference engine working on data in order to retrieve knowledge;
- *Proof*: inference validation engine. As there is no unanimously killer app in the field of knowledge inference, some validation must be performed in order to validate outputs from the *Logic* layer;
- *Trust*: mechanism providing the suitable level of security of the shared data by means of cryptography with *Digital Signatures*.

To these layers, Berners-Lee added in [BL06] a *User Interface & Applications* layer. Figure 3.2 illustrates its SW architecture presented in 2006. It can be seen that the development of SW standards such as OWL are now part of the overall architecture. The *Rule Interchange Format* (RIF) [GHMP06] aims at developing standard methodologies to support existing rule-based methodology interoperability.

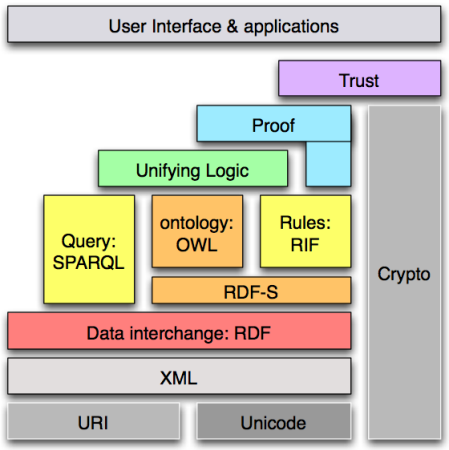


Figure 3.2 • *The adapted Semantic Web layered structure*

3.3 Data structure

The *lingua franca* for data structure and storage in the SW is incontestably the RDF. Firstly, a formal definition of an RDF graph will be proposed. Secondly, an overview of the stack in which the RDF integrates itself will be introduced. Finally, the choice of this particular model among the variety of existing ones will be vindicated.

3.3.1 Definitions

We have seen that RDF stands for *Resource Description Framework*. It was created as an additional layer to XML in 1999 [Tau06] and is commonly defined as (based on [Tim01]):

DEFINITION **RDF**
Is a framework used for defining and sharing metadata by means of graphs. Such semantic graphs enable information sharing between versatile sources.

More precisely, an RDF[5] graph can be defined according to [FHVB04, GHM04] as:

[5] Specifications of the RDF and other useful material are available in [RDF04].

DEFINITION

RDF graph

- *Is a set of triples, also called statements, of the form* $\langle$ *subject predicate object* $\rangle$*. The predicate is a directed arc going from the subject to the object. It can be seen as a property linking the object to its subject.*
- *Let us define:*
  - U*: an infinite set of RDF URI references*
  - $B = \{N_i : i \in \mathbb{N}\}$*: a set of RDF blank nodes*
  - L*: an infinite set of RDF literals (e.g. string, int, . . .)*
- *A triple*
$$(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$$
where s *is a subject,* p *a predicate and* o *an object is called an RDF triple.*
- *A graph*
$$G \subseteq (U \cup B) \times U \times (U \cup B \cup L)$$
is an RDF graph.
- *The universe of the graph* G*,*
$$univ(G)$$
is the set of elements of U*,*B *and* L *which occur in the triples of* G*.*
- *The vocabulary of the graph* G*,*
$$vocab(G) = univ(G) \cap (U \cup L)$$
is the universe of the graph where blank nodes are ignored.

An example of an RDF graph is presented in figure 3.3. The ontology used to build it is the *Friend Of A Friend* (FOAF) ontology described in [BM05].

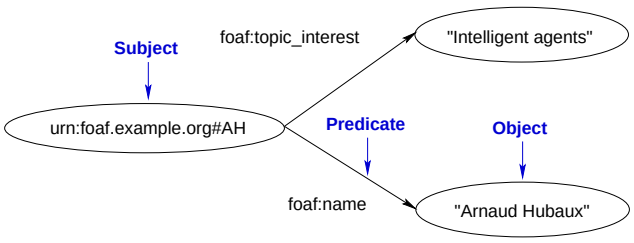


Figure 3.3 • *Example of an RDF graph based on the FOAF ontology*

More advanced considerations on RDF formal definition may be found in [ACK+01]. [AGGPE05] underlines that strictly speaking, RDF graphs are to be considered as bipartite graphs derived from 3-order hypergraphs as presented in [HG04]. Indeed, RDF graphs cannot be considered as *simple* graphs given that an edge can also be a node.

The question of the RDF physical storage has not been addressed so far. The reason is there is no killer app in that field and the choice of the tool depends much on the purpose of the stored graph. However, what is important to know is that RDF graphs are stored in *triple stores* (TS) whose implementations can vary widely. TS most popular tools will be compared in section 7.3 while 3Store will be studied more carefully.

3.3.2 RDF stack

The RDF integrates itself in a wider context as can be seen in figure 3.4, adapted from [Gar]. It illustrates the levels of constraints and representations available to build RDF graphs which will be discussed hereafter.

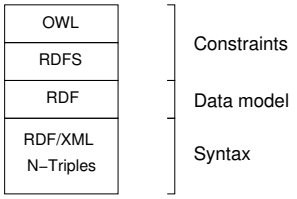


Figure 3.4 • *The RDF stack*

This powerful combination composes the core of the SW as its enables *knowledge* management and representation.

OWL

OWL technical part has been treated at point 2.2.6 and will not be developed any longer. Nevertheless, the choice of OWL as the SW standard deserves some considerations.

As previously mentioned, OWL relies on formerly defined technologies, which is shown in figure 3.5. Consequently, OWL results from the gathering of both syntactical and semantic models which enabled its creation.

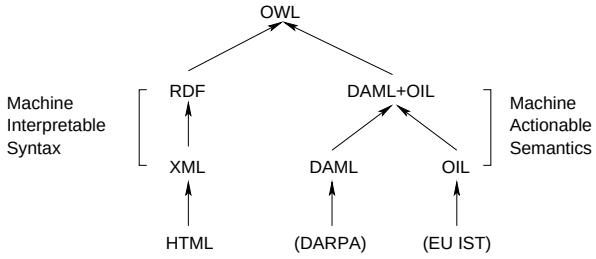


Figure 3.5 • *OWL's historical evolution*

Moreover, [Pol04] puts forward some interesting arguments promoting OWL as the most challenging SW ontology definition standard. Table 3.1 summarizes the main ideas presented in his work.

Besides those arguments, OWL has become a W3C recommendation since February 2004, which reinforces its position as a web standard and its acceptance by industrialists[6].

RDFS

As well as XML, RDF comes along with its own schema definition language named *RDF Schema* (RDFS). It enables the definition of basic constructs such as classes, subclasses, properties, subproperties, property domain and ranges, ... Simple inferences on data can then be performed, e.g. derive transitive relations. To some extent, it can be considered as a *weak* ontology specification language compared to OWL.

[6] The OWL FAQ [Hen04] answers general questions about OWL which are worth consulting.

Table 3.1 • *Summary of OWL properties*

Property	Characteristic	Benefits
Semantics are loosely coupled	*Ontologies schema representations are independent of application code and RDF models.* *Models are easily stored/referenced in loosely-coupled repositories.*	*Semantics support an evolutionary network model for exchanging data meanings and business rules.* *Semantics may be easily federated.* *Semantics may be loosely-coupled to instance data.*
Semantics are machine-actionable	*Ontologies' syntax (not graphical) is grounded in XML and RDF.* *Ontologies use consistent, standard schema semantics.* *Ontologies support well-scoped classes, properties, instances and relationships.*	*Parsers, modellers, reasoners and transformers are available.* *DL guarantees decidability and computational completeness.*
More expressive	*A powerful set of properties and class definition elements are available.* *Not just hierarchies or taxonomies can be represented (↔ XML).* *Multiple inheritance.*	*More closely models the real world.* *Axioms may be used to model rules directly into the model.*
More precise	*Relationships are atomic and unambiguous.* *Attribute overriding is disallowed.* *DL enforces consistency.* *Within a context, semantics can be unambiguous.*	*Reasoners can accommodate unknown data.* *Both explicit and implicit facts are available via queries.*

RDF/XML, N-triples

If RDF enables the creation of graph-based structures, it needs some kind of syntax-processable representation. The two most common forms are:

- *RDF/XML*: represents an RDF graph by means of a classical XML tree-like structure. This technique has the main advantage that XML is widely spread and that parsers are available in many programing languages.
- *N-Triples*: represents an RDF graph by its corresponding set of triples. This form offers a less intuitive and readable representation but has the advantage of avoiding the overload added by XML tags.

3.3.3 Vindication

Possibly the most obvious question at that point is: why the RDF in particular ? Indeed, we have seen right above that RDF graphs can be expressed by means of XML. So why define a

new data standard ? The main issue with XML is that the same content can be expressed in several ways. Let us consider the example presented in [BL98] which is illustrated in figures ranging from 3.6(a) to 3.6(c).

```
1 <author>
2       <uri>page</uri>
3       <name>Ora</name>
4 </author>
```
(a)

```
1 <document href="page">
2     <author>Ora</author>
3 </document>
```
(b)

```
1 <document>
2     <details>
3       <uri>href="page"</uri>
4       <author>
5           <name>Ora</name>
6       </author>
7       </details>
8 </document>
```
(c)

Figure 3.6 • *Meaning-equivalent XML examples*

These examples are syntactically different though they are expressing the same thing, i.e. they share common semantics. Some argue [WGA05a, BL98] that the RDF aims at filling the semantic weaknesses of XML. They claim that XML's primary function is to describe *data* whereas RDF's primary function if to describe *knowledge*.

XML Schema design only enables the settlement of data structures. If altered, such schema could prevent previously recorded data from being consistent with newly inserted data. RDF comes with definitions of ontologies by means of RDFS and OWL. Their definitions enable much more flexibility regarding structure extensions and evolutions.

Unlike namespaces used in XML, which aims at grouping related concepts[7], URI deployed in RDF graphs imply the retrievability of the associated ontology. This is a direct consequence of the use of RDF as an SW standard. It enables the sharing of knowledge which entails the free access to its content and above all to its structure. From this fundamental distinction arises the location issue. XML is limited to the enclosing document whereas RDF is not concerned with the *physical location* of data. What makes this assertion possible is the use of URI that releases the conditions of one closed single document. Any element of an RDF document can thus be addressed from anywhere.

Besides making away location boundaries, URI highly facilitates graph merges. The semantic specification of RDF graphs implies that no duplicate node can be inserted. This entails that shared nodes will only be present once in the resulting graph and predicates from both graphs will point to the same resources. Consequently, data integration from various sources and queries on merged data is by far easier than if it had to be performed on merged XML files. In other words, RDF can be considered as monotonic given that *new statements neither change nor negate the validity of previous assertions* [WGA05a].

Figure 3.7, adapted from [Her07], summarizes well what RDF proposes, i.e. a storage-independent data structure providing a high level representation of the knowledge being described and shared. Some interesting technologies appearing in this figure deserve some attention.

The *SPARQL Protocol And RDF Query Language* (SPARQL) is defined according to [SPA06] as:

[7] *Grouping related concepts* must be regarded here as a way of assigning unique prefixes in order to identify elements belonging to the same "concepts".

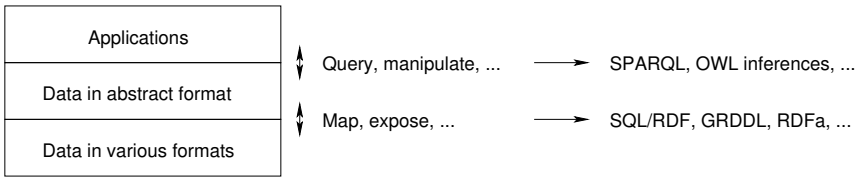


Figure 3.7 • *Illustration of data handling primitives according to its representation levels*

DEFINITION

SPARQL

Is a query language for getting information from . . . RDF graphs. It provides facilities to:

- *extract information in the form of URIs, blank nodes and literals;*
- *extract RDF subgraphs;*
- *construct new RDF graphs based on information in the queried graphs.*

As a data access language, it is suitable for both local and remote use.

Web pages contain a significant amount of structured data. However, they do not allow automatic information extraction. The *RDF in attribute* (RDFa) described in [AB07] enables it and is defined as:

DEFINITION

RDFa

Is a syntax that expresses this structured data using a set of elements and attributes that embed RDF in HTML. An important goal of RDFa is to achieve this RDF embedding without repeating existing HTML content when that content is the structured data. RDFa is designed to work with different XML dialects, e.g. XHTML1, SVG, etc., given proper schema additions. In addition, RDFa is defined so as to be compatible with non-XML HTML.

Figure 3.8 shows a typical Web page while figure 3.9 illustrates the same page where RDFa attributes have been added[8].

Gleaning Resource Descriptions from Dialects of Languages (GRDDL) described in [GRD07] favours the spreading of RDF and is defined as:

DEFINITION

GRDDL

Introduces markup based on existing standards for declaring that an XML document includes data compatible with the Resource Description Framework (RDF) and for linking to algorithms (typically represented in XSLT), for extracting this data from the document.

In other words, GRDDL provides mechanisms to extract RDF from XML and XHTML documents using transformations characteristically expressed in XSLT. Figure 3.10 shows the content of an XHTML Web page containing RDFa tags while figure 3.11 illustrates the RDF conversion obtained by the application of the transformation script defined in `RDFa2RDFXML.xsl`[9].

[8] This example was taken from [AB07].

[9] This example was taken from `http://www-sop.inria.fr/acacia/personnel/Fabien.Gandon/tmp/grddl/rdfaprimer/PrimerRDFaSection.html`. URLs have been cropped for readability. Please refer to the original page for the unaltered version.

```
<html>
    <head><title>Jo's Blog</title></head>
    <body>
...
    <p>
        I'm giving a talk at the XTech Conference about web widgets,
  on May 8th at 10am.
    </p>
...
    <p class="contactinfo">
        My name is Jo Smith. I'm a distinguished web engineer
        at
        <a href="http://example.org">
            Example.org
        </a>.
        You can contact me
        <a href="mailto:jo@example.org">
            via email
        </a>.
    </p>
...
    </body>
</html>
```

Figure 3.8 • *HTML document before the introduction of RDFa fields*

3.4 Discussion

The following observations are based on [BL06] which wisely analyzes the position of the SW on the Internet, knowledge management systems and artificial intelligence. Advanced concerns may be found by exploring further through the reference.

From the previous sections arises that the combination of agents and ontology creation languages implies resorting to AI techniques. However, it does neither mean that SW is AI nor AI is SW. The SW is a huge project whereas AI is a field of activity. Nevertheless, the SW can be a great tool to give AI projects a wider dimension by providing them with improved interoperability methods.

In his presentation Berners-Lee also underlines the following myths about the SW:

- *the SW technology is DL*:
  No, OWL is an SW language based on DL. Other languages may be preferable to OWL as they offer different expressiveness. However, they should guarantee as much interoperability as possible.

- *the SW is just about public data*:
  No, the SW project promotes knowledge sharing which does not necessarily entail its public exposure. SW techniques may also be valuable to some private and protected usage.

- *the SW is metadata for classifying documents*:
  No, document classification is only a small subset of what the SW may be used for.

- *the SW is about hand-annotated web pages*:
  No, as well as for document classification, hand-annotated web pages are far from being the mainstay of the SW.

```
<html xmlns:cal="http://www.w3.org/2002/12/cal/ical#"
      xmlns:contact="http://www.w3.org/2001/vcard-rdf/3.0#">
...
    <p class="cal:Vevent" about="#xtech_conference_talk">
        I'm giving
        <span property="cal:summary">
            a talk at the XTech Conference about web widgets
        </span>,
        on
        <span property="cal:dtstart" content="20070508T1000+0200">
            May 8th at 10am
        </span>.
    </p>
...
    <p class="contactinfo" about="http://example.org/staff/jo">
        My name is
        <span property="contact:fn">
            Jo Smith
        </span>.
        I'm a
        <span property="contact:title">
            distinguished web engineer
        </span>
        at
        <a rel="contact:org" href="http://example.org">
            Example.org
        </a>.
        You can contact me
        <a rel="contact:email" href="mailto:jo@example.org">
            via email
        </a>.
    </p>
...
```

Figure 3.9 • *HTML document with embedded RDFa fields*

- *the SW is mainly about content extracted from text* :
  No, much work still needs to be performed in order to go from one to another. Basically, the SW puts forward a language gathering relational data and logic concepts.

- *the SW is about making one big ontology*:
  No, the SW relies on a collection of interconnected ontologies.

- *the SW ontologies must all be consistent*:
  No, only the grouped ontologies working together must be consistent.

Another interesting view of the current state of the SW is presented in [SHBL06]. The remaining of this section will discuss the points underlined in the last part of this article[10] before outlining SW ethical related concerns.

Languages and standards have no value if not considered by the community of users, or at least potential users. In other words, the *uptake* is the key stage to any project to reach the acceptance phase and leave its embryonic state. However, a massive uptake is not always conceivable especially because the SW project is very ambitious. A simple look at the lowest level of the knowledge representation stack, i.e. the RDF shows that the support materials

[10]The first parts put forward current tools and applications used in the SW and investigate its current status.

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:base="http://www.dc...rdf_sem.html"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:dc="http://purl.org/dc/elements/1.1/"
      xmlns:foaf="http://xmlns.com/foaf/0.1/" >
  <head profile="http://www.w3.org/2003/g/data-view">
    <link rel="transformation" href="http://.../RDFa2RDFXML.xsl"/>
    <title>Biblio description</title>
  </head>
  <body>
    <h1>Biblio description</h1>
    <dl about="http://.../REC-rdf-mt-20040210/">
      <dt>Title</dt>
       <dd property="dc:title">
         RDF Semantics - W3C Recommendation 10 February 2004
      </dd>
      <dt>Author</dt>
       <dd rel="dc:creator" href="#a1">
        <span id="a1">
          <link rel="rdf:type" href="[foaf:Person]" />
          <span property="foaf:name">Patrick Hayes</span>
            see
            <a rel="foaf:homepage" href="http://...ID=42">homepage</a>
        </span>
       </dd>
    </dl>
  </body>
</html>
```

Figure 3.10 • *XHTML document containing RDFa tags*

are not present yet. The targeted concern here is the management of URI references present in graphs. Indeed, what could be the interest of building such graphs if their core elements were simply *unavailable* ?

Moreover, ontologies are, most probably, living structures. They are very likely to be subjected to regular updates or more major upgrades. Such changes must not prevent them from being consistent or usable. As a result, not every domain would be able to equally benefit from the SW as maintenance and development cost could vary and results not be worth the investments. In addition, the size of the community resorting to defined ontologies harshly influences their actual benefits. A possible solution to this issue is to specify two levels of ontologies, i.e. *deep* and *shallow* ontologies.

Deep ontologies intend to describe very complex sets of properties. The effort needed to set them up may be significant but the data being described is really complex. This kind of ontology is dedicated to fields like engineering, medicine or biology. *Shallow* ontologies specify somewhat unchanging terms that organize very large amounts of data, e.g. customers, accounts, stocks, ... The development effort is based on much simpler sets of elements and relations, which enhances their reusability.

Another kind of emerging ontology is *folksonomy*. It arose from community of users willing to tag information in order to classify it. This can take the form of special tags added to Web page contents. Applications exploiting that kind of tagging developed by decentralized communities are sometimes called *Web 2.0* or *social software*. However, strictly speaking folksonomies do not have to be mixed up with ontologies. Ontologies are carefully defined to avoid ambiguity, based on URIs and subject to logical inferences. Folksonomies

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:h="http://www.w3.org/1999/xhtml">
  <rdf:Description rdf:about="http://www.dc..rdf_sem.html">
    <transformation xmlns="http://www.w3.org/1999/xhtml"
        rdf:resource="http://.../RDFa2RDFXML.xsl"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://.../REC-rdf-mt-20040210/">
    <dc:title xmlns:dc="http://purl.org/dc/elements/1.1/">
      RDF Semantics - W3C Recommendation 10 February 2004
    </dc:title>
  </rdf:Description>
  <rdf:Description rdf:about="http://.../REC-rdf-mt-20040210/">
    <dc:creator xmlns:dc="http://purl.org/dc/elements/1.1/"
        rdf:resource="http://www.dc.../rdf_sem.html#a1"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.dc.../rdf_sem.html#a1">
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.dc.../rdf_sem.html#a1">
    <foaf:name xmlns:foaf="http://xmlns.com/foaf/0.1/">
      Patrick Hayes
    </foaf:name>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.dc.../rdf_sem.html#a1">
    <foaf:homepage xmlns:foaf="http://xmlns.com/foaf/0.1/"
        rdf:resource="http://...ID=42"/>
  </rdf:Description>
</rdf:RDF>
```

Figure 3.11 • *RDF document obtained after GRDDL transformation*

are loosely defined, based on words and classified by means of statistical techniques such as clustering.

Therefrom comes out that choosing the right level of ontology, if we consider folksonomy as a loose subclass of ontology, is a tricky business. Moreover, the addressed population for a given ontology must be chosen very carefully to avoid massive rejection. For the SW to be widely accepted and used this *linked information space* must encourage availability, reuse, adaptation and discovery of related knowledge.

Seductive though this approach may appear, it raises concerns about queries, mapping between ontologies, efficient browsing, and trust and provenance[11] of contents. The trust we put in the published and retrieved information may be somewhat relative. The huge scale of this project involves, for example, much more important consequences regarding false information spreading. Note that it is outside the scope of current legislations [Sha06].

Besides being a challenging technical project the SW is also socially ambitious as it involves the whole community of computer users. The following points list some of the major issues related to its deployment and only intend to awake the reader's awareness about them.

- *All traffic should be treated equally* [RC07]:
  The SW is developing tools aiming at filtering the information content by means of metadata. This could entail discriminations about the content produced on the Internet.

- *The SW should not damage public discourse* [RC07]:

[11] By provenance [SHBL06] means *the when, where and conditions under which data originated.*

The filters applied could prevent people from being heard, by means of forums for instance, and reinforce people's insularity.

- *Machines should be programmed to categorize and assign values to information* [RC07]:
  Privacy is a major concern in current information systems. The SW should prevent private information from being as easily found and processed as reviews of commercial contents.

- *The informed consent should be a central aspect of the SW* [Sha06]:
  The very high level of interrelation between information could entail unexpected information from being retrieved. As a result, SW users must be fully aware of how the information they provide could be used. Moreover, once information has been provided, stopping its spreading is almost impossible. Such dangers must be present in every user's mind.

Those ethical concerns could also prevent the uptake of the SW. Achievements in technology, even if technically perfect, must fit in a political, legal and social environment and have to pass this barrier to be fully accepted.

4 Agent System Engineering

Being busy does not always mean real work. The object of all work is production or accomplishment and to either of these ends there must be forethought, system, planning, intelligence, and honest purpose, as well as perspiration. Seeming to do is not doing.

Thomas Alva Edison

OVERVIEW

The previous chapter describing the SW project put together ontologies and agents. However, it focused mainly on ontologies as they are the basis of knowledge description. This chapter will concentrate on the engineering of complex agent systems. In the first place, a general classification of the three existing engineering approaches will be presented. In the second place, some of the most widely used methods will be discussed. We will overview how these methodologies cover the development process and at what kind of system they are targeted. The last section will synthetize them and discuss the limitations of agent system engineering.

4.1 Classification

The introduction of agents in software systems entails notably more context awareness, mobility, reactivity and distribution capabilities. High-quality engineering methods and tools are required to have a new technology accepted by industrial software developers. Even though CASE tools are the keystone to methodology acceptance, they do not help to understand them. Consequently, this chapter will mainly focus on engineering methodologies. If we refer to [SS03], a methodology can be defined as:

DEFINITION **Methodology**
Is the entire set of guidelines and activities: a full lifecycle process; a comprehensive set of concepts and models; a full set of techniques (rules, guidelines, heuristics); a fully delineated set of deliverables; a modeling language; a set of metrics; quality insurance; coding (and other) standards; reuse advice; guidelines for project management.

Current agent-oriented methodologies are either:

1. *knowledge-oriented*: suitable for cases where huge ontologies are to be managed, like in SW applications;
2. *agent-oriented*: suitable for cases where agent properties and aspects are to be precisely set up;
3. *object-oriented*: extend and adapt OO techniques to the agent paradigm.

The following sections will detail some of the leading methodologies and pinpoint how complete they are regarding the above proposed definition.

4.2 Knowledge-oriented approaches

Knowledge-Based Systems (KBS) development has always been a major concern in software design. As the concept of agent is increasingly widespreading in modern applications, KBS methodologies attempt at extending their scope to encompass agent system development. We will introduce CommonKADS and one of its agent extensions MAS-CommonKADS. Another common extension not being described is CoMoMAS [Gla96].

4.2.1 CommonKADS

CommonKADS can be considered as one of the most influential knowledge management frameworks [LAD04]. It enables the representation of specifications in textual and graphic forms. [SWdH+94] defines it as:

DEFINITION **CommonKADS**
Is a methodology dedicated to KBS projects development. It provides a set of engineering models of problem solving behaviour in its concrete organization and application context. This modeling concerns not only expert knowledge, but also the various characteristics of how that knowledge is embedded and used in the organizational environment.

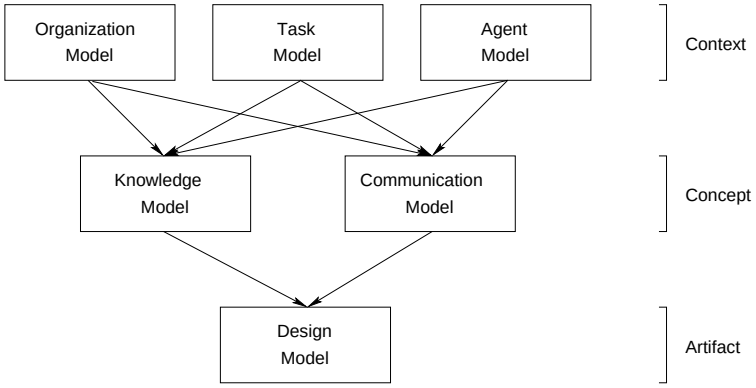


Figure 4.1 • *The CommonKADS components and layers*

The CommonKADS architecture is divided into three layers, each composed of one or more models. It only focuses on the analysis phase of the software engineering lifecycle. Figure 4.1 depicts more precisely how these components are positioned in the architecture.

The context layer is composed of the:

- *Organization model*: describes the organizational context, including the stakeholders, in which the KBS will be developed. More precisely, the stakeholders are:
  - *knowledge providers*: the domain experts;
  - *knowledge users*: the people needing the knowledge to work;
  - *knowledge decision makers*: the people in charge of decision making. They influence knowledge users' and providers' work.
- *Task model*: describes the tasks intended to fulfill the organization's goals. Each task is described by inputs, outputs, goals, control, features, environmental constraints (e.g. performance) and capabilities. Note that these properties are similar to the properties of agents we gave in chapter 1;
- *Agent model*: describes the features of the agents executing the tasks, i.e. their `name`, `type` (human, agent, software), `subclass`, `position` and `group`.

The concept layer is composed of the:

- *Knowledge/Expertise model*: describes the capabilities of agents. This model is divided into:
  - *domain knowledge*: the knowledge about the application domain;
  - *inference knowledge*: the steps needed by the reasoning system to handle domain knowledge;
  - *task knowledge*: the knowledge used to manage goals and their subdivision into subgoals and subtasks;
- *Communication model*: describes how the communication between agents works in order to achieve transactions. Note that CommonKADS was initially dedicated to human/computer interactions which entail no real support of complex interactions between agents;

The artifact layer is composed of the *Design model* which describes the software to be built through a three-stage transformation process:

1. *application design*: functional and object-oriented decomposition and constraint definitions;
2. *architectural design*: architecture design and choice of the computational infrastructure implementing it;
3. *platform design*: definition of knowledge representations and inference techniques that will be implemented.

4.2.2 MAS-CommonKADS

MAS-CommonKADS [IGGV96] extends the CommonKADS components described above in the following ways:

- *Agent model*: is extended with the following features:
  - *service*: special facilities offered to dedicated agents;
  - *goals*: agent's objectives;
  - *reasoning capabilities*: mechanism enabling the achievements of the required tasks;
  - *general capabilities*: skills (sensors, effectors, ...), agent (communication) languages, ...
  - *constraints*: norms, preferences, permissions, ...
- *Coordination model*: is a new model created to fulfill the weakness of the *Communication model*. It improves the request of service or information, the update of information, the choice of agent capabilities in transactions with other agents and it defines efficient inter-agent protocols.

MAS-CommonKADS supports the conceptualization, analysis, design, coding, testing, integration, operation and maintenance phases of the software engineering lifecycle.

4.3 Agent-oriented approaches

Approaches inherited from KBS and adapted to AOP bridge the gap between legacy methods and new technologies. However, agent-specific methodologies are now available and gaining importance in the software engineering world. Moreover, they enable the representation of the high level entities that are agents without resorting to established OO techniques. One of the most promising methodologies is Gaia which will be introduced in the next point.

4.3.1 Gaia

It is defined according to [WJK00] as:

DEFINITION **Gaia**
Is a methodology for agent-oriented analysis and design. The Gaia methodology is both general, in that it is applicable to a wide range of multi-agent systems, and comprehensive, in that it deals with both the macro-level (societal) and the micro-level (agent) aspects of systems. Gaia is founded on the view of a multi-agent system as a computational organization consisting of various interacting roles.

The Gaia methodology is dedicated to the development of large-scale applications where:

- agents are computational systems endowed with much computational resources;
- agents cooperate in order to primarily achieve system goals and not individual goals;
- agents are implemented in heterogeneous programming languages;
- agent interactions are static. They do not change at run-time;
- agent abilities and provided services are static. They do not change at run-time.

Figure 4.2 depicts the different layers and components of the Gaia methodology.

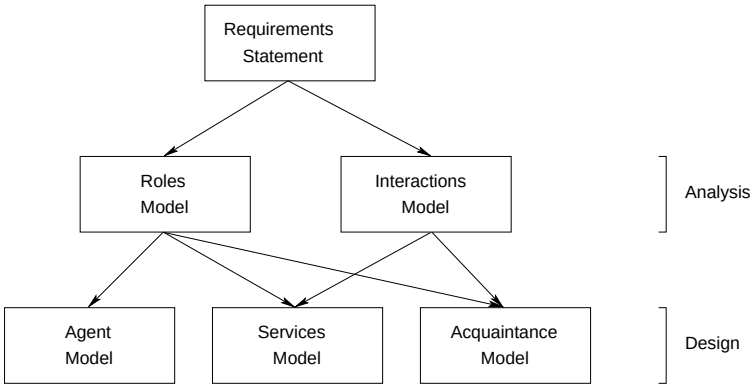


Figure 4.2 • *The Gaia components and layers*

The requirement phase aims at systematically capturing the requirements of the system to be developed and should be detailed enough to enable their direct implementation. However, they do not depend on the implementation framework employed in the following phases.

The analysis phase intends to build a complete understanding of the system and its structure. The system organization being captured is a *collection of roles, that stand in certain relationships to one another, and that take part in systematic, institutionalized patterns of interactions with other roles.* Its components are the:

- *Roles model*: describes the key roles in the system. The roles define abstract descriptions of the expected functions of the entities. Permissions/rights and responsabilities are associated with each role;
- *Interactions model*: describes the relations between the various roles. Dedicated protocols are associated with these interactions to model inter-role communication.

The design phase intends to transform the model produced during the analysis phase into a low-level abstraction model which might be implemented by more traditional design techniques such as OO. Note that the Gaia methodology does not specify how the agents achieve their tasks. Its components are the:

- *Agent model*: describes the needed agent types and their run-time instances. There can be a one-to-one relation between agent types and roles. However, several roles can be grouped and assigned to an agent type;

- *Service model*: describes the services associated to the agent roles and the properties of those services. A service can be seen as a function associated to an agent;
- *Acquaintance model*: describes the communication links between the agent types. They do not define the contents of messages or their sent time, they only define communication paths. It is derived from the agent and interaction models.

Note that an extension of Gaia is the *Role Oriented Analysis and Design for Multi-Agent Programming* (ROADMAP) methodology described in [JPS02]. It adds elements to deal with requirement analysis and resorts to AUML to give more precise interaction definitions.

4.4 Object-oriented approaches

A convenient way to promote AOP as a valuable paradigm is to present it as an extension of well established techniques such as OO. We have seen however that agents are higher level entities than objects. As a result, the direct application of OO techniques to agent systems engineering is not suitable. The most common engineering methodologies extending them are MaSE, MESSAGE, Prometheus and Tropos, which will be discussed hereafter. The last point will shortly describe AUML which is not, strictly speaking, a methodology.

4.4.1 MaSE

In order to define the *Multiagent Systems Engineering* (MaSE) methodology [DeL99] refers to the six challenges of MAS put forward by Sycara in [Syc98]:

1. *how to decompose problems and allocate tasks to individual agents ?*
2. *how to coordinate agent control and communications ?*
3. *how to make multiple agents act in a coherent manner ?*
4. *how to make individual agents reason about other agents and the state of coordination ?*
5. *how to reconcile conflicting goals between coordinating agents ?*
6. *how to engineer practical multiagent systems ?*

DEFINITION

MaSE

Is a methodology whose goals are:

1. *to engineer practical systems, and to provide a framework for solving the first five challenges. It uses the abstraction provided by multiagent systems for developing intelligent, distributed software systems;*
2. *to define a methodology specifically for formal agent system synthesis.*

The first goal is achieved by creating two languages:

1. *Agent Modelling Language*: provides a graphical notation to represent agents in a system and their interfaces to other agents;
2. *Agent Definition Language*: relies on the first order predicate logic and intends to describe the internal behaviour of each individual agent.

To meet the second goal, a formal semantics will be defined for both languages.

Moreover, beside being implementation independent, MaSE offers the ability to track changes made during every step of the development process. Figure 4.3 [WD00] sketches the layers and design components constituting the MaSE methodology. Note that it iterates across all phases in order to add details to the models.

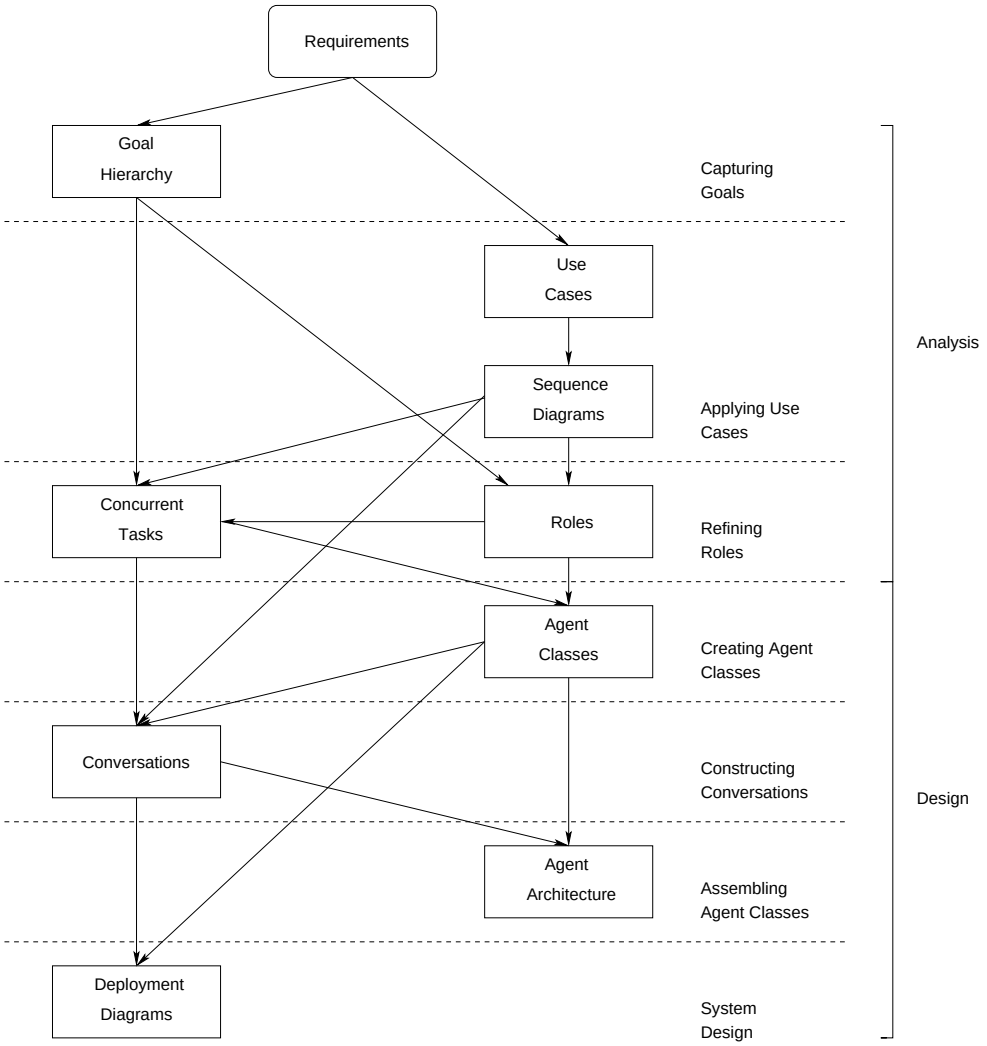


Figure 4.3 • *The MaSE components and layers*

The aims of each of the phases are:

- *Capturing goals*: intends to define a goal hierarchy diagram from the initial system specifications. Levels are determined by their importance degree, are functionally related to their parents and same level goals are roughly equal in scope;

- *Applying use cases*: eases the transition from specifications and goals to agents. Typically, at least one sequence diagram is created for each use case;

- *Refining roles*: enables the creation of role models. Roles synthesize what is expressed in the goal hierarchy and sequence diagrams;
- *Creating agent classes*: identifies agent classes from role models to create an agent class diagram, which also contains conversations between them;
- *Constructing conversations*: defines the conversation protocols between pairs of agents. Each conversation is defined by two communication class diagrams for the initiator and responder;
- *Assembling agent classes*: creates the content of the agent classes. Content architecture might be based on templates such as those defined at point 1.3[1];
- *System design*: instantiates agent classes as actual agents. They are represented in a deployment diagram in order to define the actual number, types and locations of the agents within the system.

A tool called *agentTool* supporting the MaSE methodology is available at [Lab]. Note that to simplify the research, MaSE puts some limitations:

- the system is closed and all external interfaces are encapsulated by an agent that participates in the system communication protocol;
- the system is static, i.e. agents cannot be created, destroyed or moved at run-time;
- inter-agent communication is only one-to-one, i.e. there is no multicast.

4.4.2 MESSAGE

The *Methodology for Engineering Systems of Software Agents* (MESSAGE), also referred to as MESSAGE/UML, is defined according to [CCG+02] as:

DEFINITION **MESSAGE**
Is a methodology that covers MAS analysis and design and is intended for use in mainstream software engineering departments. It extends UML by contributing agent knowledge level concepts, and diagrams with notations for viewing them.

MESSAGE methodology is based on the combination of the best features taken from *AUML*, *Gaia*, *CommonKADS* and goal-analysis techniques.

The micro level entities are modelled by means of UML. The combination of all these models form the macro level model of the system. The entities being modelled are called *knowledge level entities* and belong to one of those classes:

- *concrete entities*: include agents, organizations (groups of agents), roles (describe the characteristics of an agent in a particular context) and resources (e.g. databases);
- *activities*: are composed of tasks and interactions;
- *mental state entities*: define goals (associate an agent with a situation);
- *information*: describes objects containing chunks of information;

[1] Note that in practice the *constructing conversations* and *assembling agent classes* are closely coupled. MaSE recommends iterating between both phases and developing them as simultaneously as possible.

- *messages*: are objects communicated between agents.

For reason of brevity, they will not be developed any further. More details are available in [CCG+02].

The analysis model is based on five views or models representing the system to build:

- *organizational view*: shows the concrete entities present in the system and their relationships;
- *goal/task view*: shows the goals, tasks and situations. Situations take the form of attributes of goals and tasks. They are used to represent logical dependencies to form graphs representing goal decompositions, task achieving goals, ...;
- *agent/role view*: shows agents and roles. Diagrams represent agents/roles and the goals they are responsible for, resources they control, ...;
- *interaction view*: shows interactions between agents/roles, i.e. initiators, information passed, ...;
- *domain view*: shows domain concepts and how they are interrelated.

Each of these models are successively refined in *levels*, each of them representing a more detailed view of the system to be built.

4.4.3 Prometheus

The Prometheus methodology is defined according to [PW02] as:

DEFINITION **Prometheus**
Is a detailed and complete methodology covering all activities required in developing intelligent agent systems.

It claims to differ from competing methodologies in that it:

- supports the development of intelligent agents whose architecture is based on goals, belief, desires, plans and events;
- provides a complete engineering lifecycle and a detailed process;
- has been used by both industrial practitioners and undergraduate students;
- allows design to be performed at multiple levels of abstraction;
- uses an iterative process over the software engineering phases;
- provides automatable cross checking of design artifacts.

The methodology follows a three-phase approach depicted in figure 4.4. The intent of those iteratively run phases are the:

- *System specification*: captures the system inputs and outputs. Inputs are called *percepts* and environment effectors are called *actions*. Note that *percepts* are not *events*. *Events* are significant occurrences for the agent system whereas *percepts* are raw data available to the agent. System functionalities[2] describing what it should do and how it processes inputs to provide outputs must also be described. As functionalities focus on particular parts of the system, use cases are developed to offer more holistic views;

[2] Note that a *functionality* in the Prometheus methodology is referred to as a *role* in the other methodologies.

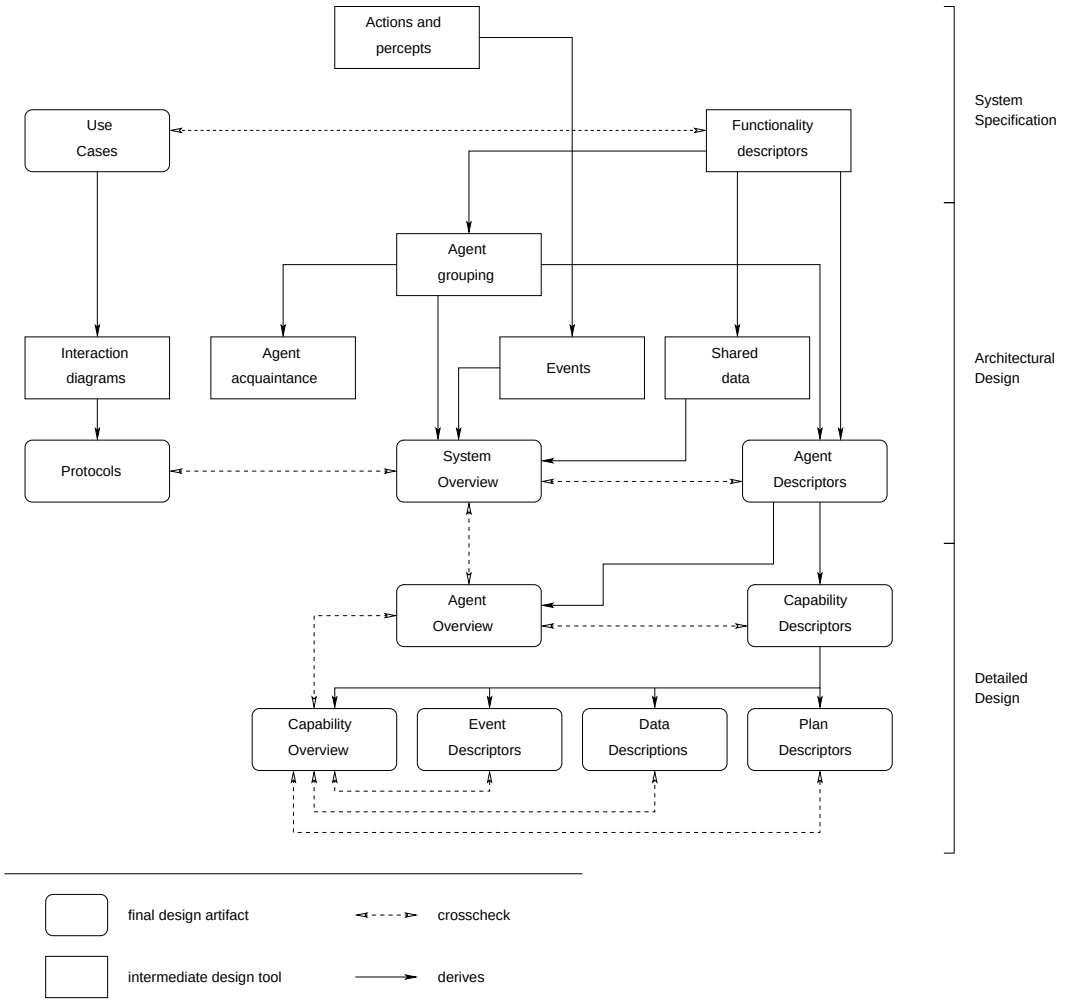


Figure 4.4 • *The Prometheus components and layers*

- *Architectural design*: determines which agents should be built and how they will interact. An agent encompasses one or more of the functionalities previously defined. Functionality selection intends to maximize coherence and minimize coupling with other agents. Acquaintance models are drawn to assess agent coupling. Once the agents have been identified, agent descriptors can be established (number, lifetime, I/O, ...). Events resulting from environment *percept* processing must be clearly identified. Data shared by agents has to be clarified and reduced as much as possible. The system overview diagram can then be defined. It is the keystone of this development stage as it gathers agents, events and shared data. Interaction diagrams are built to present inter-agent communication and are extended by interaction protocol specifying communication sequences;

- *Detailed design*: focuses on the agents' internal details (capabilities, internal events, plans, data structures) and on their task achievements within the system. A platform supporting AOP must be chosen to implement them (e.g. *JACK*). The result of this

detailed analysis are agent overview diagrams showing how these elements are linked to each other. Capability descriptors are specified for events, data and plans. Note that capabilities can be imbricated.

To these elements, Prometheus adds a *data dictionary* which should be completed and checked at every step of the development process.

4.4.4 Tropos

Tropos intends to develop a real AOP methodology in that it does not only resort to adapting UML diagrams to agent modelling. It argues that agent applications are conceived at knowledge level and the use of UML forces the translation of mentalistic notions into software level notions (e.g. classes, methods, attributes, ...). As a result, agent specific notions must be reintroduced in the programming phase. The Tropos methodology is defined according to [BGG+04] as:

DEFINITION

Tropos
Is a methodology based on two key ideas:

1. *the notion of agent and all related mentalistic notions (for instance goals and plans) are used in all phases of software development, from early analysis down to the actual implementation;*
2. *the very early phases of requirement analysis are thoroughly covered.*

As it covers the whole software engineering lifecycle, Tropos aims at modelling the system to build and its environment by incrementally improving the designed models. It follows five development phases. The first one comes from the requirement engineering field and is based on the i^* model [Yu97] which is:

DEFINITION

i*
Is a framework developed for modelling and reasoning about organizational environments and their information systems. It consists of two main modelling components. The strategic dependency model is used to describe the dependency relationships among various actors in an organizational context. The strategic rationale model is used to describe stakeholder interests and concerns, and how they might be addressed by various configurations of systems and environments.

The last four phases are widely supported in the software engineering field.

- *Early requirements*: capture domain stakeholders and model them as social actors. They depend on each other for goals to be achieved, plans to be performed and resources to be provided. They aim at modelling the system's *why's* beside the *how* and *what*;
- *Late requirements*: extend the conceptual models developed during the early requirement stage by including new actors representing the system and their relationships with actors of the environment. All the functional and non-functional requirements of the future system are expressed through the dependencies;
- *Architectural design*: specifies the system's global architecture in terms of sub-systems (actors) interconnected through data and control flows (inter-actor dependencies). System actors are also mapped to software agents which are characterized by specific capabilities;

- *Detailed Design*: defines agent capabilities and interactions. At this stage, an implementation platform (e.g. JACK) should have been selected. The detailed design developed with this framework has to map directly to the code to be produced;
- *Implementation*: implements the results of the previous design phases in a fully functional system.

Modelling focuses on actors, dependency, goals, plan and capability. Actor and dependency modelling is achieved through *actor diagrams*. Goals and plans are modelled by means of *goal diagrams* following three reasoning techniques. *Means-end* analysis intends to identify the plans, resources and softgoals which provide means for achieving goals. *Contribution analysis* precises how existing goals contribute, positively or negatively, to the goal being analyzed. *AND/OR decomposition* decomposes a given goal into subgoals following an AND/OR notation. Note that all the models produced so far rely on the i^* framework. Capability modelling is performed by means of *capability* and *plan diagrams* using i^* notations, UML activity diagrams and AUML interaction diagrams.

4.4.5 AUML

Agent UML (AUML) aims at bridging the gap between current UML dedicated to OO application design and AOP. It is defined according to [OPB00] as:

DEFINITION **AUML**
Considers agents as an extension of active objects, exhibiting both dynamic autonomy (the ability to initiate action without external invocation) and deterministic autonomy (the ability to refuse or modify an external request). Its view of agents as the next step beyond objects leads it to develop extensions to UML and idioms within UML to accommodate the distinctive requirements of agents.

AUML proposes a three-layer approach to represent *agent interaction protocols* (AIPs), which is:

DEFINITION **Agent Interaction Protocol**
Describes a communication pattern as an allowed sequence of messages between agents and the constraints on the content of those messages.

Levels are respectively representing the:

1. *overall protocol*: enables the modelling of the interaction protocol as a whole. This entails easier pattern definition for future reuse in other systems. The most suitable UML protocol definition techniques are:
   - *packages*: conceptually group classes in order to form relatively independent subsystems. Interactions intra/inter packages can be modelled by means of sequence diagrams. An example is sketched in figure 4.5;
   - *templates*: enable package customization. They are represented as boxes superposed on the generic package. A template is divided into three categories which are role parameters, constraints and communication acts (CA). An example is shown in figure 4.6;
2. *interactions among agents*: focus on UML dynamic models and extend them in the following ways:

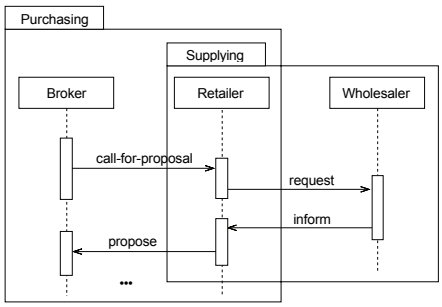


Figure 4.5 • *Example of AIP modelling with packages*

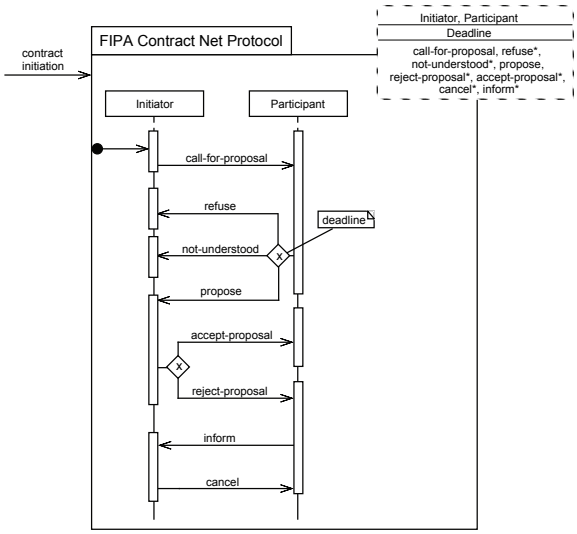


Figure 4.6 • *Example of AIP modelling with templates*

- *sequence diagrams*: extend their members by naming them according to the pattern `agent_name/role:class`. Competing thread modelization is also encouraged and method call arrows should be labelled with CA;
- *collaboration diagrams*: are extended similarly to sequence diagrams. It is argued that the view they propose, even if semantically similar, may sometimes be more easily adapted;
- *activity diagrams*: are promoted as they provide an explicit thread of control which is very useful to model complex interaction protocols involving a lot of competing processing;
- *statecharts*: are primarily used to model constraints on the protocol as the view presented is interaction-centred rather than agent-centred. Note that they are not meant to be implemented as agents. Alternatively, they can be integrated in roles which will be played by agents;

3. *internal agent processing*: can be easily modelled by means of activity diagrams and statecharts. They should both be coupled with sequence diagrams to specify the behaviours associated to every CA.

A textual notation and a tool used for AIP design are proposed in [Win05]. Other extensions to UML such as agent role changes, agent mobility or the use of agents as interfaces between components are also discussed in [OPB00]. Note that the AUML working group activities have not evolved much since late 2004.

4.5 Synthesis

The above sections presented the three main classes of AOP methodologies. In order to have a global view of these leading methodologies, we will synthesize the key concepts put forward in each of them. Figure 4.7, adapted from [SBPL04], shows the software engineering lifecycle phases they cover.

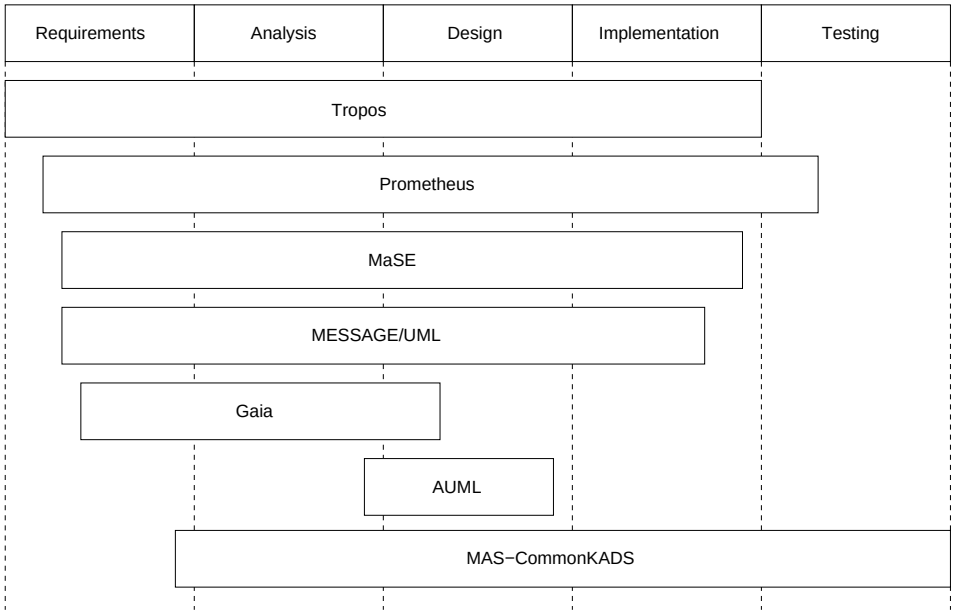


Figure 4.7 • *Synthesis of the engineering lifecycle phases*

Table 4.1, taken from [LAD04], summarizes what we addressed in the previous sections. AUML is not part of it as we do not consider it as an actual methodology. Note that some methodologies took some elements from *agile* methods and more particularly from the *Rational Unified Process* (RUP) which is defined according to [ASRW02] as :

DEFINITION

RUP

Is an iterative approach for OO systems, and it strongly embraces use cases for modelling requirements and building the foundation for a system. It is inclined towards OO development. It does not implicitly rule out other methods, although the proposed modelling method, UML, is particulary suited for OO development.

Table 4.1 • *Summary of agent system engineering methodologies*

Approach	Basis	Application area	Agency support
CommonKADS MAS-CommonKADS	KM	Knowledge centred applications	Organization tasks, agents, knowledge, interaction
Gaia	AO	Coarse-grained computational systems	Roles, agents, knowledge, interaction, services, acquaintance
MaSE	OO and RUP	Heterogeneous MAS	Goals, roles, interaction, agents
MESSAGE	OO and RUP	Coarse-grained computational systems	Organizations, goals, tasks, agents, roles, knowledge, interaction
Prometheus	OO and BDI	BDI agents	Goals, belief, plans, events, agents, interaction, capabilities
Tropos	OO and BDI	BDI agents	Actor, goals, plan, resource, capability, interaction

The interested reader should consult [DW03] and [SS03] which propose detailed evaluation frameworks for agent-oriented methodologies. They take MaSE, Prometheus, Tropos and Gaia as case studies for their framework evaluation and present the obtained results.

4.6 Discussion

This section will attempt to attract system developers' attention about some possible flaws of AOP. Attractive though agent systems are, they must be well thought and carefully designed. We will present here the work done in [JW99] which describes agent-oriented software engineering principles as well as their pitfalls.

It might first be difficult to maintain a balance between the reactive and proactive capabilities of agents. They may indeed give priority to more immediate tasks and never be able to achieve their objectives. On the opposite, they may only attempt to achieve their goals without taking care of their environment evolution. Furthermore, these capabilities could rapidly entail unpredictable behaviours, possibly amplified by the asynchronous nature of the events being generated. Consequently, as an agent might depend on others to achieve its tasks, their completion could be compromised. The last issue is the *notion of emergent behaviour* resulting from the interaction and composition with other entities. A group phenomenon could occur and direct individual entity behaviours, just like in human communities.

In addition to those inherent threats, some common pitfalls require particular attention:

- *you oversell agent solutions, or fail to understand where agents may usefully be applied*;
- *you get religious or dogmatic about agents*;
- *you don't know why you want agents*;

- *you want to build generic solutions to one-off problems*;
- *you believe that agents are a silver bullet*;
- *you forget you are developing software*;
- *you forget you are developing multi-threaded software*;
- *your design doesn't exploit concurrency*;
- *you decide you want your own agent architecture*;
- *your agents use too much AI*;
- *you see agents everywhere*;
- *you have too few agents*;
- *you spend all your time implementing infrastructure*;
- *your agents interact too freely or in an unorganized way.*

Nevertheless, these arguments should not discourage or prevent software developers from resorting to agents. Agents as well as objects or any other paradigm have their dark sides. Agents system engineering is neither more error-prone nor difficult than any other one. It is simply newer and consequently less mature and tested.

Part II

Sharing System

5 System Presention

Everything should be made as simple as possible – but no simpler.

Albert Einstein

OVERVIEW

So as to define the developed system, it is necessary to precise its context as well as the functions it intends to take on. This chapter will, in the first place, introduce the project framework and the engineering methodology followed. In the second place, the concepts needed to understand the system built will be underlined. Finally, an abstract description of the system and its position in the environment will be put forward as well as a characteristic scenario.

5.1 Introduction

The work presented here is partially supported by the European Commission under the 6th Framework Program through the *Content Analysis and Retrieval Technologies to Apply Knowledge Extraction to massive Recording* (CARETAKER) project (Activity: Semantic-based Knowledge and Content Systems, contract no.:FP6-027231). *It aims at studying, developing and assessing multimedia knowledge-based content analysis, knowledge extraction components, and metadata management sub-systems in the context of automated situation awareness, diagnosis and decision support*[1]. It was also supported by the Multitel research centre in Mons[2].

The following points will introduce the system objectives and the selected development methodology. Since several aspects of this project are interrelated, many cross-references will be used throughout the following chapters to illustrate the coupling between both agents and knowledge storage technologies.

5.1.1 Objectives

The addressed part of the CARETAKER project was defined as follows. *The event recognition subsystem will serve as a demonstrator applying the technology in a real-time system of between 10 and 20 cameras and microphones*[3]. *The purpose of this small-scale, on-line, prototype system is to demonstrate both the effectiveness of the technology, and how the implementation can scale to the massive recording scenarios of real-scale monitoring systems. To enable the results of WP4 and WP5 to be used in a real-scale environment with an abundance of sensors, state-of-the-art open standards for distributed event driven processing will be used.*

In order to bridge the gap between end-user interfaces and stored knowledge, a Web server will be coupled to the system. Note however that the system we built only offers an interface to it by means of queries.

As this project aims at promoting new technologies, the design of an innovative system was required. The adaptation of technologies used in the SW and AI came out as conspicuous. Consequently, AOP and SW data structures were the keystone of this part of the CARETAKER project.

The system we built has been presented at the IS&T/SPIE 19th annual symposium on electronic imaging science and technology in [LHC+07].

5.1.2 Methodology

Chapter 4 put forward some of the most common agent system engineering methodologies. The one followed during the development of this project is Prometheus. In order to justify this choice, we will refer to section 4.5 which compared every of them.

Note that the intent of this project is not to stick to a particular methodology. It aims more at taking advantage of the flexibility of the agent approach than at building a perfect agent system. Consequently, the selection of the methodology was made to define a general development framework rather than strict guidelines. Moreover, the following chapters detailing the system will not strictly follow the Prometheus methodology. They will simply underline to which parts they relate.

[1] For further information about the CARETAKER project, please visit `http://www.ist-caretaker.org/`

[2] For further information, please consult `http://www.multitel.be/`.

[3] Note that no information about audio processing was available during development stages. This side of the project has thus not been considered.

Why Prometheus ?

A point not mentioned so far is that no clear requirements have been formulated. Moreover, data specifications were neither settled before nor after the system development. As a result the mandatory features were adaptability, flexibility, capability of evolution and fast prototype creation.

As Prometheus focuses notably on the implementation and testing phases it is well suited for concrete system conception. Furthermore, it enables the representation of agent specific concepts such as plans, events, ... in a very intuitive way.

It is also supported by the JACK toolkit [Age06] which is:

DEFINITION **JACK**
Is an agent-oriented development environment built on top of and integrated with the Java programming language. It includes all components of the Java development environment as well as offering specific extensions to implement agent behaviour.

It was of great help in every step of the design, implementation and debugging steps. Moreover, the detailed design phase results were automatically converted into components needed during the implementation phase, which sharply reduces coding efforts. The tracing tool was additionally of great help during debugging phases.

Why not other methodologies ?

CommonKADS does not offer real implementation techniques and was thus not an acceptable choice. Given the short available development time, well mastered techniques based on OO were required to guarantee faster operability. Consequently, its extension MAS-CommonKDAS has not been considered as an option.

Gaia is not much more adapted as it does not address implementation and testing issues.

Although Tropos covers most of the development lifecycle phases it attaches much importance to the requirement analysis. As it was not a major concern in this case, this methodology has been set aside.

MESSAGE does not seem, to our knowledge, to be supported by significant tools and was not retained for this reason.

MaSE has not been chosen because it did not seem to be widely used. However, it is supported by a rather advanced tool, *agentTool*, which could be worth some attention.

5.2 Definitions

In order to define the system, we first need to precise the way data will be transferred, stored and their types. The following points will address those issues. However, more technical definitions about used concepts will come in the following chapters.

5.2.1 Data transfer

Data will be transferred from one entity to another by means of a telecommunication network. The possible sources emitting on the netwok are:

- *sensors*. They are typically video cameras, microphones, ... and transmit data over an IP channel;

- *system*. It transfers low/mid-level analysis and replies to queries by means of an RSS flow;
- *video content analysers*. They produce two kinds of results based on low/mid-level and high-level analysis, which will be defined right below. Analysers in charge of the high-level analysis will also be able to perform queries on previous analysis results and receive replies to them. Video analysis results as well as queries and replies will be transported by means of an RSS flow;
- *web servers*. They can perform queries on analysis results and receive replies to them. Queries and replies will be carried by means of an RSS flow.

The three commonly admitted levels used to divide image representations are the *low*, *mid* and *high*-levels. They all intend to identify different elements of the pictures. These representations can be defined as following:

DEFINITION **Low-level representation**
Is composed of atomic or primitive events that can be extracted directly from the raw sensory signals. Usually, this representation aims at identifying primitive events that can be localized both in time and (image-)space such as motion detection, background subtraction, edge and texture based recognition of moving objects, ...

DEFINITION **Mid-level representation**
Deals with the object tracking (and identification) task and activity characterization over an extended time basis and/or over multiple sensors. Typical examples are multi-object tracking and simple activity recognition (e.g. person standing, seated, carrying luggage) in a single camera or over a network of camera.

DEFINITION **High-level representation**
Aims at recognizing more complex events, usually modelled by an expert of the application domain (the user), and expressed through a set of rules. Among the classical high-level events one can cite the "abandoned luggage detection", "red traffic light violation detection", ...

The key point is to note the increasing degrees of abstraction provided by these levels. Low-level analysis focuses on raw information present in pictures whereas high-level analysis attempts to recognize patterns or scenarios. High-level analysis may thus require information about previously recorded pictures while low-level analysis is based only on the current one. Figures 5.1, 5.2 and 5.3, taken from [LHC+07], illustrate these concepts for an abandoned luggage detection scenario.

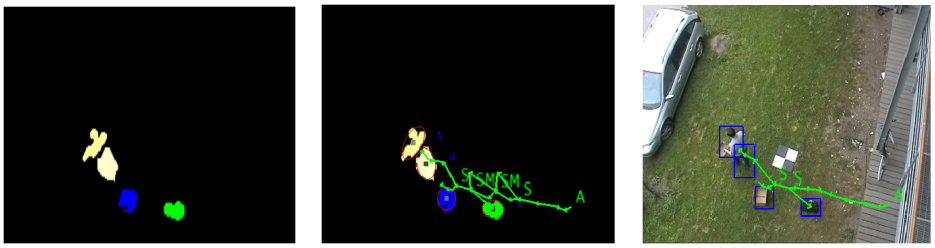
5.2.2 Data storage

All data will be stored in a unique data warehouse (DW) which can be defined according to [Inm02] as:

DEFINITION **Data warehouse**
Is a subject-oriented, integrated, nonvolatile, and time-variant collection of data in support of management decisions.

(a) Picture (b) Picture background (c) Picture foreground mask

Figure 5.1 • *Illustration of low level features identification*



(a) Blob features (b) Rough tracking results (c) Final filtered tracks

Figure 5.2 • *Illustration of mid-level features identification*

Subject-oriented means in this case that the contained data provides relevant information about a dedicated subject rather than about a company's current operations. Integrated comes from the fact that data from various sources is merged in a single coherent whole. It is also nonvolatile as it is never deleted once saved. Consequently, the gathered information always grows and gives a more relevant view of the business it relates to. Moreover, so as to identify data filling the DW, it is marked with time tags. The intent of a DW is principally to improve and facilitate the decision making in business processes. We will see in the following chapters how we have adapted this concept to our particular needs[4].

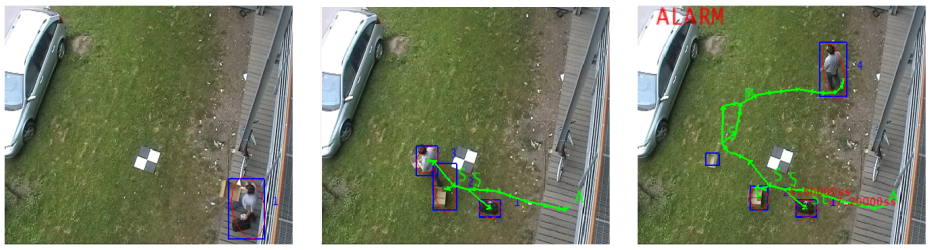
5.2.3 Data types

The data format used by sensors does not affect the system being developed and will thus not be addressed.

Data transferred over the RSS flow will be analysis results, queries and replies. They will all be formatted in XML in order to guarantee system consistency and compliance with standards. These XML documents will be wrapped into RSS feeds as shown in figure 5.4. Attention should be paid to the fact that XML documents contained in a feed have all the same type, i.e. low/mid-level analysis results, high-level analysis results, queries or replies.

The type of the data internally handled by the system is the RDF.

[4] Besides [Inm02], the interested reader should consult [Wid95] which addresses research problems and [ZGMHW95] which investigates view maintenance in DWs.



(a) Mobile object appears (b) Immobile objects stem from the mobile object (c) Immobile objects exceed maximum time and cause an alarm raise

Figure 5.3 • *Illustration of high-level features identification*

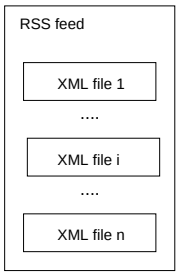


Figure 5.4 • *XML documents wrapped in an RSS feed*

5.3 Description

Firstly, a global scheme of the system and its surrounding environment will be presented. Secondly, a more detailed view of the system will be depicted and its internal work will be addressed. Finally, a characteristic scenario will be proposed to illustrate the system in action. That is however not to be considered as a use case. It only intends to help understanding the coming chapters.

5.3.1 System framework

The system framework is presented in figure 5.5. Its constituting entities are the:

- *sensors*. They are responsible for information retrieval about the physical environment;
- *signal handling systems*. They are responsible for data conversion from sensors to WP4 understandable formats;
- *WP4*. It is responsible for low/mid-level analysis from data coming from sensors;
- *WP5*. It is responsible for high-level analysis from data coming from WP4 ;
- *demonstrator*. It is the system built. It is responsible for:
  - handling analysis results;
  - handling queries and replies;
- *Web server*. It is responsible for:

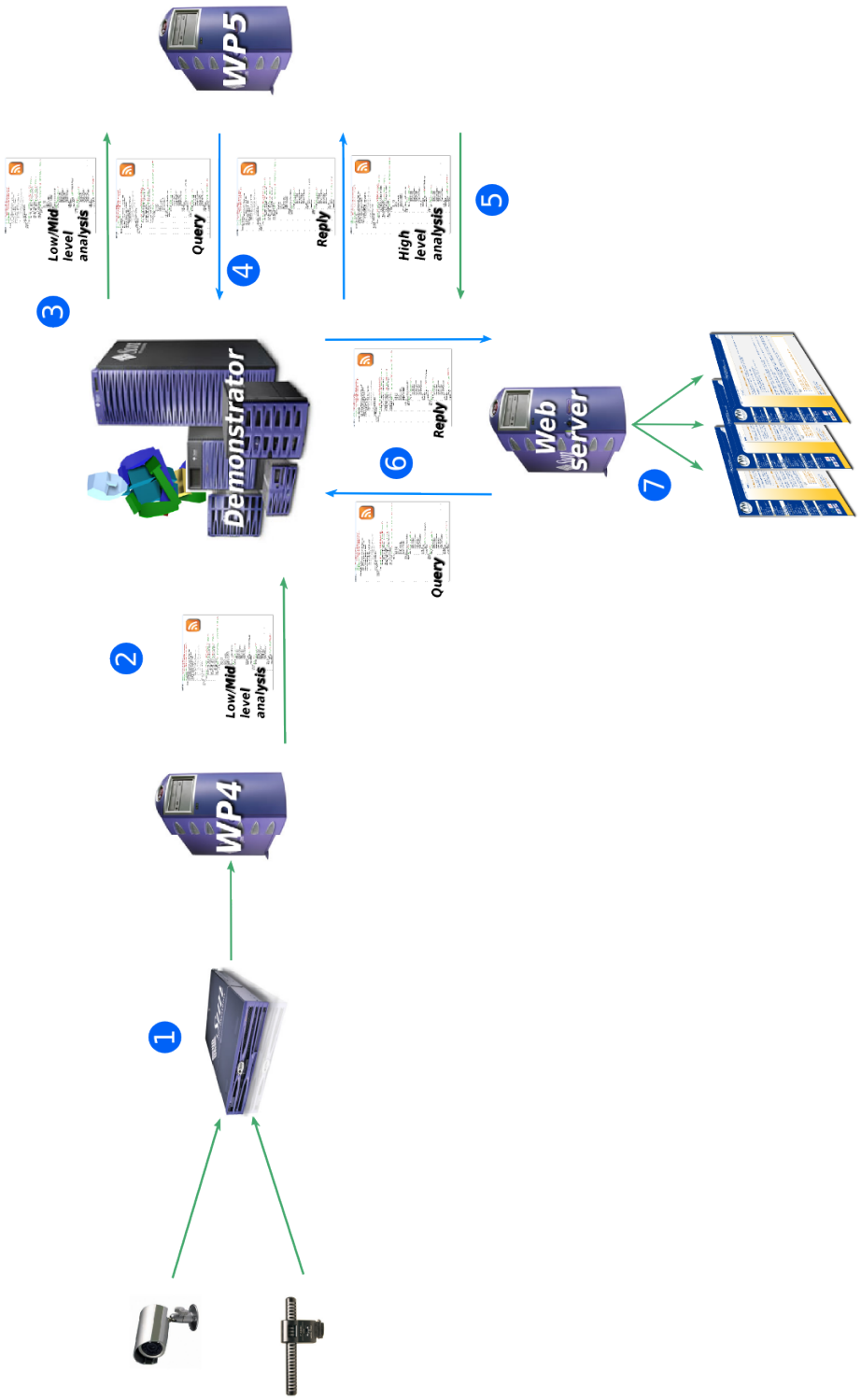


Figure 5.5 • *System framework*

– performing queries on the demonstrator. They are made by users by means of their Web interfaces;
– handling replies and updating users' current Web page.

The number of queries/replies occurring during high-level analysis or web server accesses is undetermined. The flow coming from sensors, entailing video analysis, passing through WP4, the demonstrator and WP5 must always be run in an atomic way. In other words, every sensed event triggers a thorough analysis of the system.

5.3.2 System architecture

The system abstract architecture is presented in figure 5.6. The sketched "persons" are representing agents. They will be responsible for the data flow management inside the system.

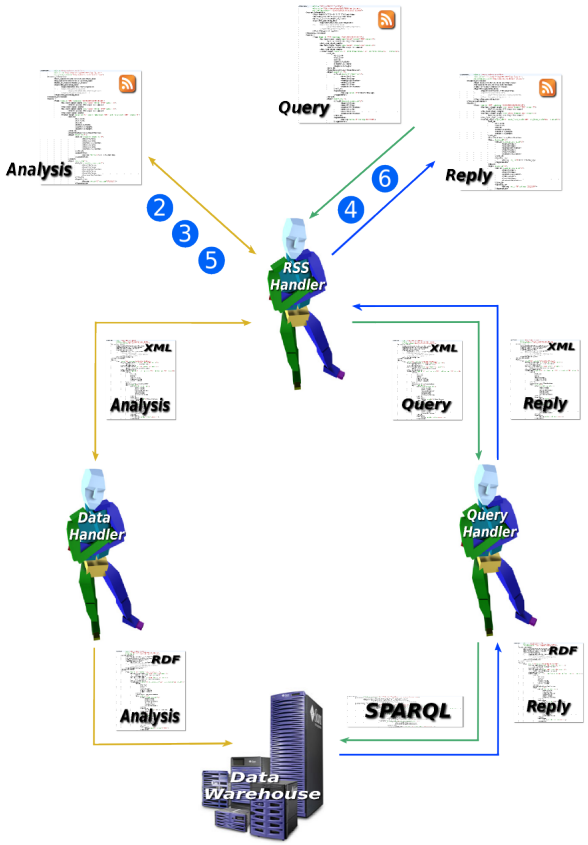


Figure 5.6 • *System abstract architecture*

The system architecture mostly shows data conversion. For low/mid/high-level video analysis, there comes:

$$RSS \rightarrow XML \rightarrow RDF$$

Whereas for low/mid-level video analysis there also comes:

$$XML \rightarrow RSS$$

This is due to the fact that low/mid-level analysis results need to be sent to WP5 for high-level analysis.

Data is stored as RDF graphs in the data warehouse. In order to perform queries on RDF graphs some conversions of XML queries and RDF replies are also required:

$$RSS \rightarrow XML \rightarrow SPARQL \rightarrow RDF \rightarrow XML \rightarrow RSS$$

Note that more than one SPARQL query can be issued for a single XML query.

5.3.3 Characteristic scenario

With the intention of ensuring full system comprehension, a typical scenario will be presented. Its content is for illustration purpose only and may slightly differ from reality. The numbers presented in figure 5.5 will be used to identify the current process part. The same numbers are present in figure 5.6 and show how external events influence the system. Note that reactions are seen as a complete sequence of actions of one single physical observation. It will be repeated as many times as observations are available. If the video sensors work on a 15 *frame per second* (FPS) basis, it will be repeated 15 times a second. Furthermore, in order to come to the analysis conclusion, more than a single observation will be needed.

Context

Let us assume:

- a room containing some restricted access equipment. Only a limited set of people have been granted access to it;
- a video camera pointing at the door. We assume here it is the single entry point;
- a voice recognition device unlocking the door for authorized people only.

Event

An unauthorized person enters the room. The voice recognition device will not report an allowed entry while pictures coming from the camera will show someone entering the room.

Reactions

1. Data coming from both sensors are handled and sent in a WP4-understandable format ❶;
2. WP4 performs a low/mid-level analysis on received data and identifies notably the open door, no access authorization from the voice recognition device and someone in the room;
3. WP4 exports its analysis results in XML;
4. WP4 wraps the XML in an RSS feed and sends it to the demonstrator ❷;
5. The demonstrator extracts the XML from the RSS feed;
6. The demonstrator converts the XML into RDF and saves it in the DW;
7. The demonstrator wraps the received XML in an RSS feed and sends it to WP5 for high-level analysis ❸;

8. WP5 extracts the XML from the RSS feed;
9. WP5 performs a high-level analysis on received data. This can require information about previously saved data. In this case:
   (a) WP5 emits a query to the demonstrator in XML format and wraps it in an RSS feed;
   (b) The demonstrator extracts the XML from the RSS feed;
   (c) The demonstrator retrieves the replies from the DW;
   (d) The demonstrator converts the RDF replies into XML and wraps them in an RSS feed;
   (e) The demonstrator sends the results to WP5;
   (f) WP5 extracts the XML replies from the RSS feed ❹;
   (g) WP5 carries on its analysis and performs other queries if needed;
10. WP5 ends its analysis and identifies the event as an `unauthorized_entry`;
11. WP5 exports its results to XML;
12. WP5 wraps the XML in an RSS feed and sends it to the demonstrator ❺;
13. The demonstrator extracts the XML from the RSS feed;
14. The demonstrator analyses the XML, extracts new information and saves it as RDF in the DW;
15. A security guard consults its Web interface and asks for `unauthorized_entry` events;
16. The web server emits a query in XML format to the demonstrator and wraps it in an RSS feed;
17. The demonstrator extracts the XML from the RSS feed;
18. The demonstrator retrieves the replies from the DW;
19. The demonstrator converts the RDF replies into XML and wraps them in an RSS feed;
20. The demonstrator sends the replies to the web server
21. The web server extracts the XML results from the RSS feed ❻;
22. The web server updates the security agent's interface ❼;
23. The security agent notices the intrusion and apprehends the intruder[5].

[5]Note that a happy end has been retained to underline the relevance and interest of the system. Current video analysis technologies still fail to guarantee fully reliable analysis results.

6 System Design

Architecture is one part science, one part craft and two parts art.

David Rutten

OVERVIEW

The previous chapter briefly introduced the system to build and its context. This mandatory part being achieved, we can now focus on the system design and on the specification of its internal components. The environment will be described in order to have a precise specification of the messages handled and their influence on the system. Then, the system architecture will be detailed. Agent functionalities will finally be addressed.

6.1 Environment

In order to establish a correct analysis of the system, its surrounding environment must be precised. Firstly, the RSS flow used to interface the system with its environment will be described. Secondly, the format of the exchanged data will be addressed.

6.1.1 RSS flow

The communication with the environment will only be achieved by means of an RSS flow. The RSS[1] technology has been selected because it:

- enables the easy transfer of data having a textual form;
- allows the content identification of transferred data;
- avoids the redefinition of a new data encapsulation language;
- does not impose any transport protocol.

The protocols used will be defined hereafter as well as interfaces between the agent system and the outside world. Note that the specification of the RSS elements used is described in section A.1.

Concepts

The RSS is a data format supported by Harvard. Several versions gave it different acronyms which are:

- RSS 0.9, RSS 1.0: *RDF Site Summary*;
- RSS 0.9*, RSS 1.0: *Rich Site Summary*;
- RSS 2.0: *Really Simple Syndication*.

Its 2.0 version, which is the one we have used, is defined according to [RSS] as:

DEFINITION **RSS**
RSS is a Web content syndication format. RSS is a dialect of XML. All RSS files must conform to the XML 1.0 specification, as published on the World Wide Web Consortium website.

Before introducing the interface between the RSS flow and the agent managing it, it is necessary to express the chosen perspective. The classical use of RSS, as depicted in figure 6.1(a), does not seem to fit the system needs. Indeed, the resource monopolization due to server polling for new events is intolerable. Even though it does not harm usual Web applications, this does not prove to be the most suitable choice for performance demanding systems.

The selected approach, depicted in figure 6.1(b), is of the type publisher/subscriber[2]. It appears less resource consuming as much from the machine point of view as from the networking one.

In our case, the used protocol will be brought back to its simpler expression directly above the TCP/IP layers. High-level protocols such as `http`, `ftp`, ... will not be considered.

[1] Note that RSS is currently challenged by *Atom* [ATO06], which is another format supported by the *Internet Engineering Task Force* (IETF).

[2] The interested reader should consult [PLJ05] that describes an advanced graph-based subscription mechanism.

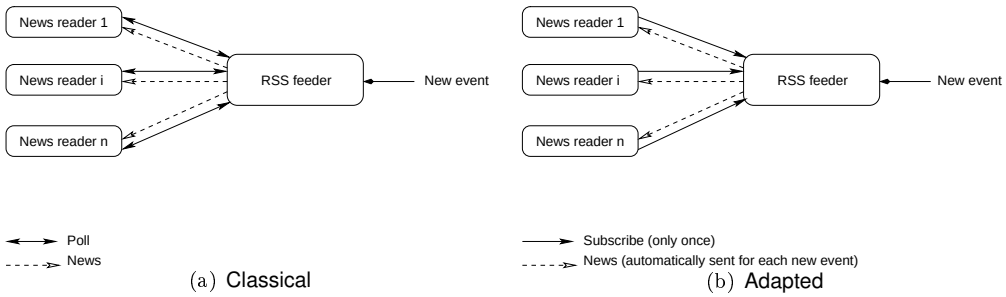


Figure 6.1 • *Comparison of different RSS protocols*

Interfaces

The RSS flow is the only entry point in the system for external data. In order to switch from objects to agents, a Java thread per connection has been set up. It is the direct interface between agents and the outside world. The interaction sequences will be presented in point 6.2.1.

6.1.2 Data schema

We saw in chapter 5 that data specification had not been defined yet. This entails strong adaptability and flexibility conditions. In order to meet these criteria, the choice performed was to pass as parameters, at system boot, the data schemas of received data[3]. The language used to express them is the *REgular LAnguage for XML Next Generation* (Relax NG, abbreviated RNG) which is defined according to [RNG03] as:

DEFINITION

Relax NG

Is a schema language for XML. The key features of RELAX NG are that it:

- *is simple;*
- *is easy to learn;*
- *has both an XML syntax and a compact non-XML syntax;*
- *does not change the information set of an XML document;*
- *supports XML namespaces*
- *treats attributes uniformly with elements so far as possible;*
- *has unrestricted support for unordered content;*
- *has unrestricted support for mixed content;*
- *has a solid theoretical basis;*
- *can partner with a separate datatyping language (such as W3C XML Schema Datatypes);*

Even if the *XML Schema Definition* (XSD) appears as the *lingua franca* of schema specification languages in the industrial world, it turns out that the users' community tends to prefer Relax NG. They appreciate its better readability and the various degrees of simplification performed in its grammar.

[3] We will see in section 7.1.1 how they are actually used

From the above proposed definition comes out that Relax NG is not an ontology definition language. An important thing to keep in mind is that the developed system is only in charge of storing analysis results and making them available. Consequently, it is not at this level that ontology specification comes into play. The two fields possibly needing ontology definitions are the AIP and video analysis. As we are using JACK, the AIP ontology is specified by the FIPA. The video analysis ontology was unfortunately not available at the time of development. Even if it did not really influence the implementation, its absence prevented any possible optimization regarding graph structures[4]. Note that some video analysis approaches, such as those presented in [DMK+05, DPM+04, HWS05], also introduce ontologies in low and mid-level analysis processes but barely investigate higher-level analysis.

6.2 Design

This section aims at presenting the interactions, i.e. the external and internal events triggering system processes. External interaction protocols will be depicted while internal interactions will be developed in the system architecture part. Agent and event descriptions will also be detailed. The data warehouse specification will be addressed in chapter 7.

6.2.1 External interactions

The external events, coming from the RSS flow, which are processed by the system and wrapped are:

1. low/mid-level video analysis results formatted in XML;
2. high-level video analysis results formatted in XML;
3. queries formatted in XML coming from the high-level video analyzer[5];
4. queries formatted in XML coming from the web server.

The external events, emitted on the RSS flow, which are generated by the system are:

1. low/mid-level video analysis results formatted in XML triggering high-level video analysis;
2. replies formatted in XML sent to high-level video analysis queries;
3. replies formatted in XML sent to web server queries.

The following interaction diagram, promoted by Prometheus, intends to model the interactions depicted hereabove. In order to reflect the agent approach, only asynchronous messages were used to avoid losing the autonomy and dynamism characterizing agents. The use of stereotypes `«plan»` and `«event»`, taken from the Prometheus terminology, has been created to avoid OO-like method calls and message passing representations.

The sequence diagram presented in figure 6.2 illustrates how the RSS input flow interfaces with the agents. It shows how an RSS feed containing an XML document is generated by an external entity, sent over the network, handled by a specific RSS feed reader and managed by the `RSSFlowHandler`. The `Outside world` corresponds to the simulator implemented during

[4] Not to mention the frustration it entailed from the curious developer.
[5] For query format specifications, please refer to section A.2.

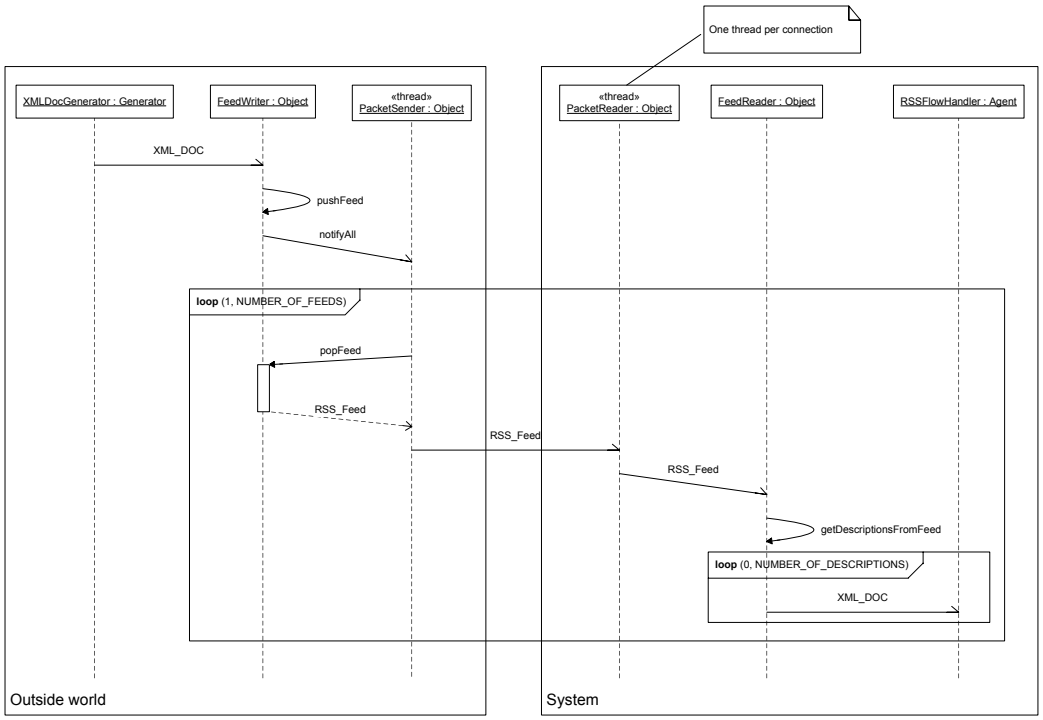


Figure 6.2 • *Sequence diagram depicting the interactions between outside world and system*

test phases. In order to offer a flexible and scalable system, one thread per connection is established. A `FeedReader` object is used to centralize and extract XML contents from RSS feeds. The contents of the descriptions are then passed to the `RSSFlowHandler`.

The sequence diagram presented in figure 6.3 illustrates how agents interface with the RSS output flow. It shows how the RSS feeds to publish are made available by the `RSSFlowHandler`. They are then sent over the network by threads assigned to each connection. In other words, only the thread associated to a given connection can access and transfer data originating from it. Once a feed is read and removed from the RSS feeds pending list, it is sent to the specified receiver.

6.2.2 Architecture

Figure 6.4 sketches the agent system and the communicated events. The following points will respectively address them. Note that events are not to be considered as agent-level events. If their contents are always semantically similar, their formats and representations within and outside the system differ. The intent here is to show the process flows triggered by external events.

Figure 6.5 presents the *agent overview diagram*. It was modelled with JACK according to the Prometheus recommendations. It shows the actual events passed among the agents. They implement the *abstract* events flows described below.

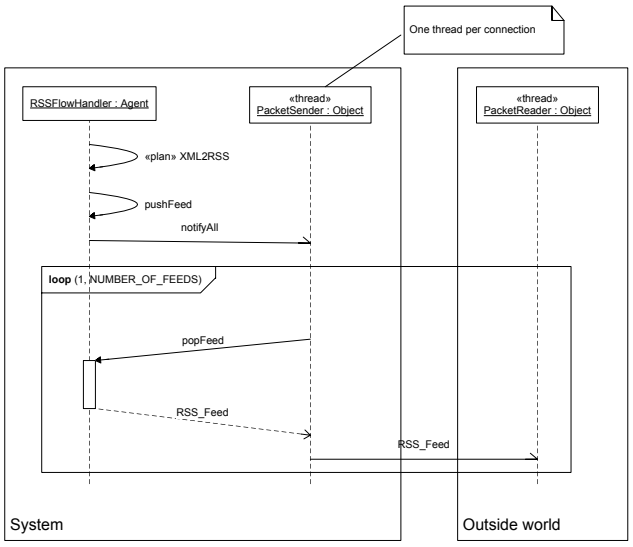


Figure 6.3 • *Sequence diagram depicting the interactions between system and outside world*

Agents

Before detailing agent functionalities it is interesting to discuss them with the definition proposed in point 1.2.2. In order to sense the environment and perform required operations the agents are able to fetch events. These events entail an interpretation and the execution of a specific action. As the flow of events is neither *a priori* known nor limited, agent lifetime must but infinite. Consequently, they are not simple processes run once and killed after task completion. Moreover, the results of the event processing also produce output events affecting the agents' environment. The concept of goal has intentionally been avoided so far. We are working in a real-time constrained environment which implies as simple and fast processes as possible. All agent goals are thus implicit: directly run the plan in charge of the event received. A goal taxonomy implying the choice of the most appropriate subgoal to fulfill was unadapted and consequently ruled out.

Another very specific kind of agent used for data handling will be introduced in chapter 7.

- `RSSFlowHandler`:
  - handles the data flow coming from the RSS publishers;
  - transmits the XML extracted from the feeds to the corresponding agents according to their types;
  - wraps and transmits XML replies to queries to their emitters;
  - wraps and transmits XML low/mid-level analysis to the high-level analyser.
- `DataHandler`:
  - transmits low/mid-level analysis to the `RSSFlowHandler` to obtain high-level analysis results;
  - identifies, from high-level analysis results, semantic links with the information already saved;

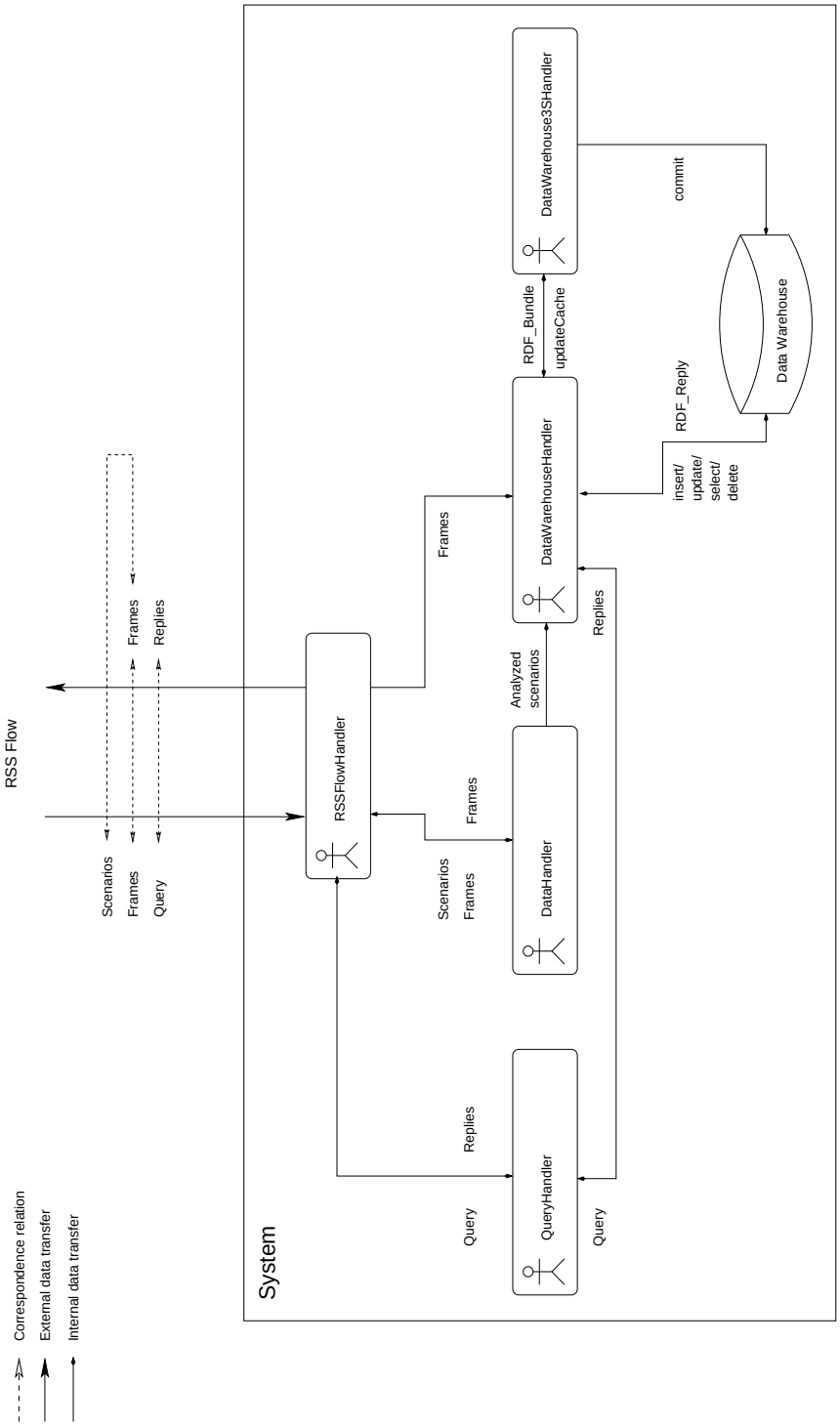


Figure 6.4 • *System architecture overview*

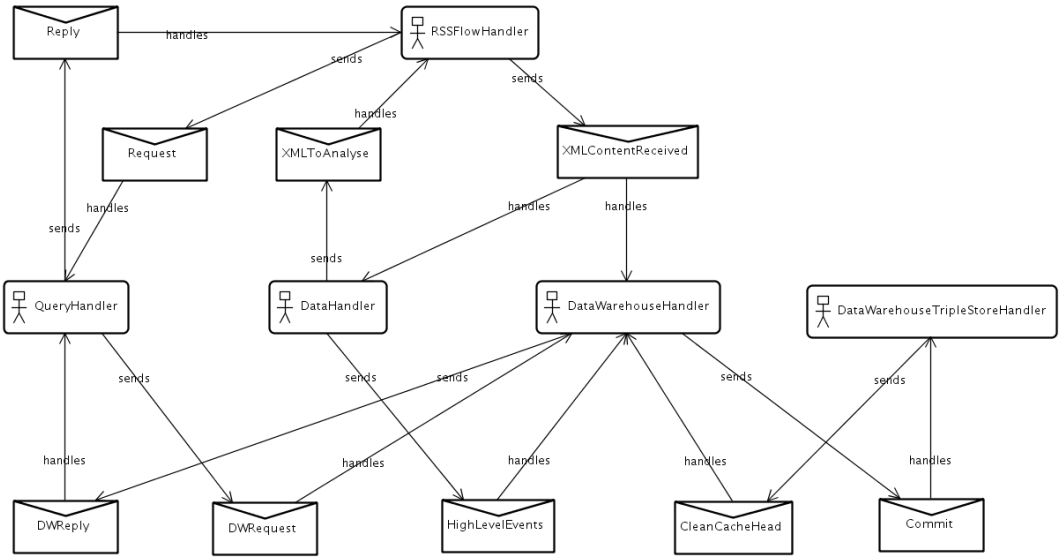


Figure 6.5 • *Agent overview diagram*

  - transmits to the `DataWarehouseHandler` the semantic links to add[6].

- `DataWarehouseHandler`:
  - saves data in cache;
  - transmits *commit* requests to the `DataWarehouse3SHandler`;
  - removes committed data from cache after *commit* if necessary;
  - run queries from the `QueryHandler` and returns results.
- `DataWarehouse3SHandler`:
  - asynchronously saves data on a non-volatile support;
  - confirms *commit* completion to the `DataWarehouseHandler`.
- `QueryHandler`:
  - handles query execution of data coming from the `RSSFlowHander`;
  - adds queries to the pending query queue;
  - transmits requests to the `DataWarehouseHandler`;
  - retrieves replies from the `DataWarehouseHandler`;
  - removes executed queries from the pending query queue;
  - transmits results to the `RSSFlowHandler`.

[6] Note that these last two functionalities were not fully implemented by lack of information about high-level analysis results.

Events

- `Scenarios`: contain the results of the high-level analysis, i.e. identified scenarios, for some given `Frames`. There can be more than one frame included in one XML result file for performance reasons;
- `Frames`: contain the results of the low/mid-level analysis, i.e. basic features directly identifiable in the picture. There can be more than one frame involved in one XML result file for performance reasons;
- `Query`: contains the query to perform. The sender's identity does not influence the query process;
- `Replies`: contain the replies to a given `Query`. There can be more than one reply to a query;
- `Analyzed scenarios`: contain the representation of the new information to be inserted in the DW;
- `RDF_Bundle`: contains a bundle of RDF subgraphs to asynchronously commit on a non volatile support;
- `commit`: contains a commit order if a certain threshold is reached. See point 6.2.3 for more details;
- `updateCache`: contains an update cache order. It implicitly means that the bundle previously sent has been committed;
- `insert`: contains an insert in DW (in cache more precisely) order;
- `update`: contains an update DW (in cache more precisely) order;
- `select`: contains a select from DW order. There can be more than one `select` for one `Query`;
- `delete`: contains a delete from cache order. It pops a certain number of AA from the cache if needed. See point 7.1.3 for more information about cache size;
- `RDF_Reply`: contains a single RDF reply corresponding to a given query.

6.2.3 Internal interactions

The next three points will depict, by means of sequence diagrams, how agents interact with each other. Note that `XML_DOC` and `XML_*` have the same content. The renaming was performed for readability reasons.

Knowledge query

The sequence diagram of knowledge query is depicted in figure 6.6. The same sequence is used for queries coming from both the high-level analyzer and the web server. The intent is to ensure simplicity and flexibility. It shows how queries coming from the `RSSFlowHandler` are put in a queue before being handled by the `DataWarehouseHandler`. Once the replies have been sent back, the query is removed from the queue and the replies are converted into XML before being wrapped in an RSS feed by the `RSSFlowHandler`.

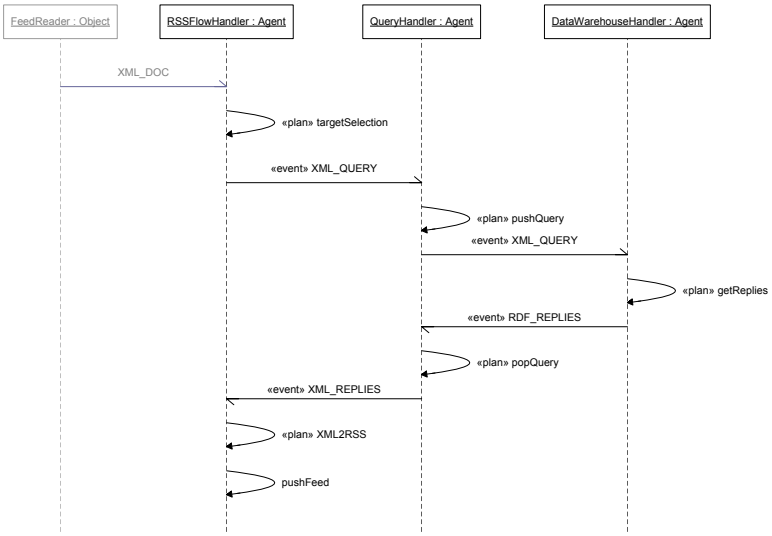


Figure 6.6 • *Sequence diagram depicting the query process*

Knowledge handling

The sequence diagram of the received low/mid-level analysis results is presented in figure 6.7. It shows how the low/mid-level analysis results are addressed to the corresponding agents. The `DataHandler` saves in a temporary memory the AA corresponding to the received document and sends a copy of the XML to the `RSSFlowHandler`. This copy is transmitted by the `RSSFlowHandler` to the high-level analyzer. The `DataWarehouseHandler` converts the XML into RDF before saving it in the DW. More precisely, it stores it in cache and triggers a data `commit` if a certain threshold is reached. This case will be discussed in point 6.2.3.

The sequence diagram of the received high-level analysis results is presented in figure 6.8. It shows how the high-level analysis result is analyzed by the `DataHandler` before being sent to the `DataWarehouseHandler`, which saves it in the DW. It is then removed from the `DataHandler`'s low/mid-level results list.

Knowledge storing

The sequence diagram of the knowledge committing in 3store and the cache update is presented in figure 6.9. It shows the sequence of actions performed when a given threshold has been reached. It is the number of files that is not directly saved on a non volatile support. Besides saving data in an asynchronous way, we noticed that storing large files in the TS improves system performances. However, beyond a certain size, performances decreased. Consequently, this threshold must be chosen and studied very carefully. During our tests, a threshold of 25 files offered satisfying performances.

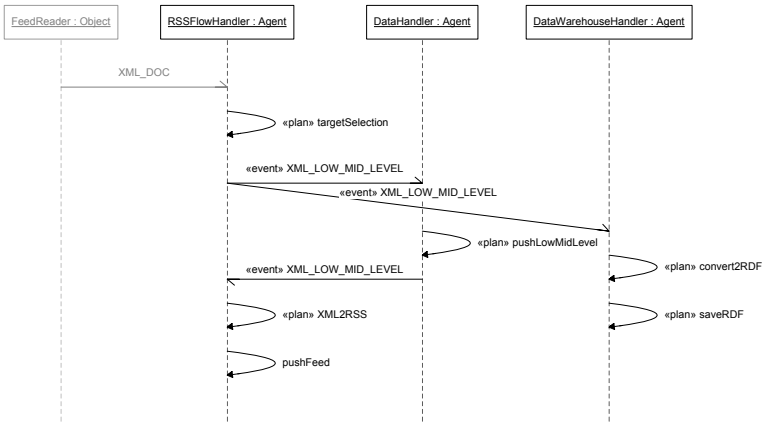


Figure 6.7 • *Sequence diagram depicting the low/mid-level analysis result process*

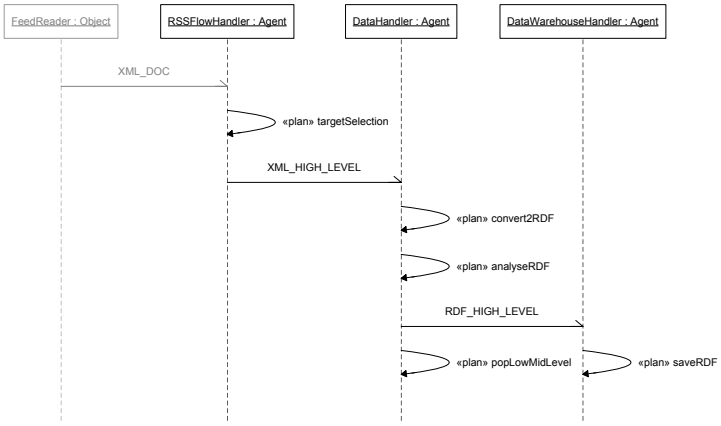


Figure 6.8 • *Sequence diagram depicting the high-level analysis result process*

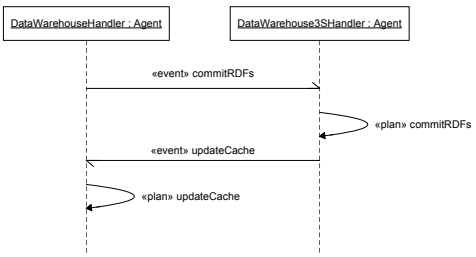


Figure 6.9 • *Sequence diagram depicting the commit process*

7 Knowledge Handling

The next best thing to knowing something is knowing where to find it.

Samuel Johnson

OVERVIEW

Now that a clear overview of the system architecture has been put forward, some attention can be given to the storage and handling of the knowledge produced by the video analysis. The composition of the DW will first be addressed. Then, the knowledge structure of the RDF graph, i.e. its schema, will be discussed as well as the way it is managed. Finally, the available TS will be screened and the 3Store selection will be vindicated.

7.1 Architecture

The overview of the DW architecture is presented in figure 7.1. An important thing to notice is that queries are performed on both cache and 3Store. Particular synchronization mechanisms, delineated in point 7.2.2, were set up to ensure results soundness and completeness. Moreover, so as to respect the nonvolatile data warehouse property, knowledge is permanently stored in 3Store by means of the `commit` operation which was presented in point 6.2.2. Note that if we refer to the Prometheus methodology, the DW is the only shared data among agents in the system.

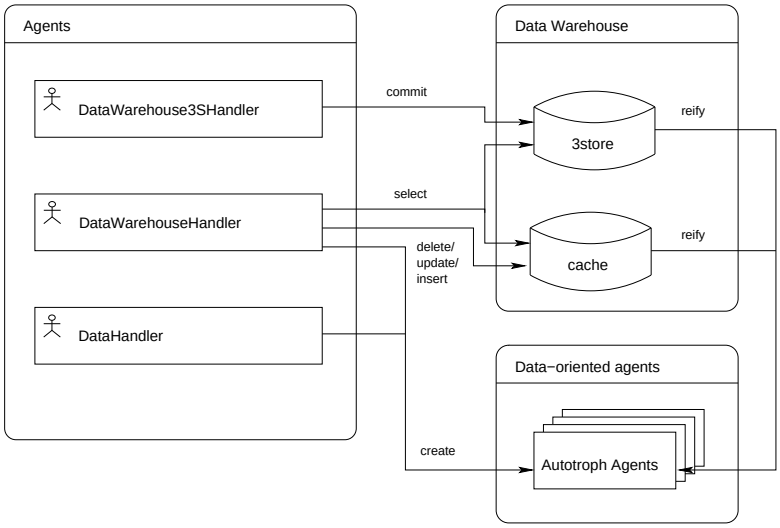


Figure 7.1 • *Data warehouse architecture overview*

The following points will detail the components sketched in figure 7.1.

7.1.1 Autotroph agents

The *Autotroph Agents* (AAs)[1] form a set of data-oriented agents incarnating raw XML documents. However, they do not have to be assimilated to simple XML document "translations". In the first place the origin of this unexpected name and the properties characterizing them will be put forward. In the second place, the questionable status of agent attributed to AA will be discussed.

If one refers to biology, autotrophs are organisms able to create organic compound, from inorganic compounds. As a result, they are always the basis of any food chain. Autotrophic replicators are thus able to reproduce themselves by building their own resources from substances whose nature is not equivalent to the ones constituting them. As we will see, the data-oriented agents we use are close to these organisms.

Beforehand, the *reification* term recurrently coming over in the remainder of this document calls for some clarification. It is defined in the field of knowledge as:

DEFINITION **Reification (Knowledge)**
Representation of facts enabling manipulations such as comparison of logical assertions.

[1] Note that AAs were implemented in pure Java and not with JACK for evident performance reasons.

whereas in computer science it is

DEFINITION **Reification (Computer science)**
Conversion of an abstract model into a more concrete and handable one in order to perform specific processing

We consider AAs as data-oriented agents rather than as simple objects for the following reasons. They are more evolved than simple data containers endowed with basic accessors such as *getters* and *setters*. Indeed, each of them is capable of:

- self-replication, i.e. to replicate its own structure and content;
- self-reification, i.e. to retrieve the knowledge stored in the DW and to instantiate its skeleton with it;
- self-exportation to XML;
- self-exportation to RDF/XML and N-Triples and optimization of the graph produced in order to minimise reification time and the required space;
- updating its components;
- modifying its skeleton from new data;
- merging with other AAs of possibly different types;
- applying a mask on the exported data[2].

The skeleton here is considered as the uninstantiated structure of the AA. Actually, a skeleton is generated for every RNG document passed at system initialization. Once constructed, a skeleton is replicated to create a new AA which can then be instantiated. This way of handling input data guarantees the needed flexibility and adaptability. Moreover, given the AA is the smallest unit handled by agent processes, this dynamic and self-managing structure really sped up and eased implementation. The internal, multiply chained, structure of an AA is depicted in figure 7.2[3]. Each element of the structure can be seen as a directly addressable sub-agent capable of the operations described above, which enables an excellent handling flexibility. One can notice the mapping between the XML and the AA tree structure. Furthermore, `element` and `attribute` types were added to reflect the similarity with the transformed XML document.

Figure 7.3(a) illustrates how the AA skeletons are created at system initialization. They are stored and accessible to an AA instantiator method which is called everytime a new XML document arrives. The type of the received XML document is analyzed and the matching skeleton is replicated. The newly created AA is then instantiated, which is depicted in figure 7.3(b).

So as to evaluate the status of the AA, we will refer to the agent definition presented in point 1.2.2. Regarding environment perception and event handling, we cannot strictly speak about real consciousness. Indeed, AAs are dormant entities not actually aware of their environment evolutions. The events they handle take the form of Java method calls that cannot, rightly to our opinion, be considered as agent-level events. Nevertheless, they do not stop running after the execution of a specific action. They are also goal driven and

[2] For full specifications of masks, please refer to section A.3.

[3] Implementation-specific variables were omitted to avoid overloading the figure with secondary information.

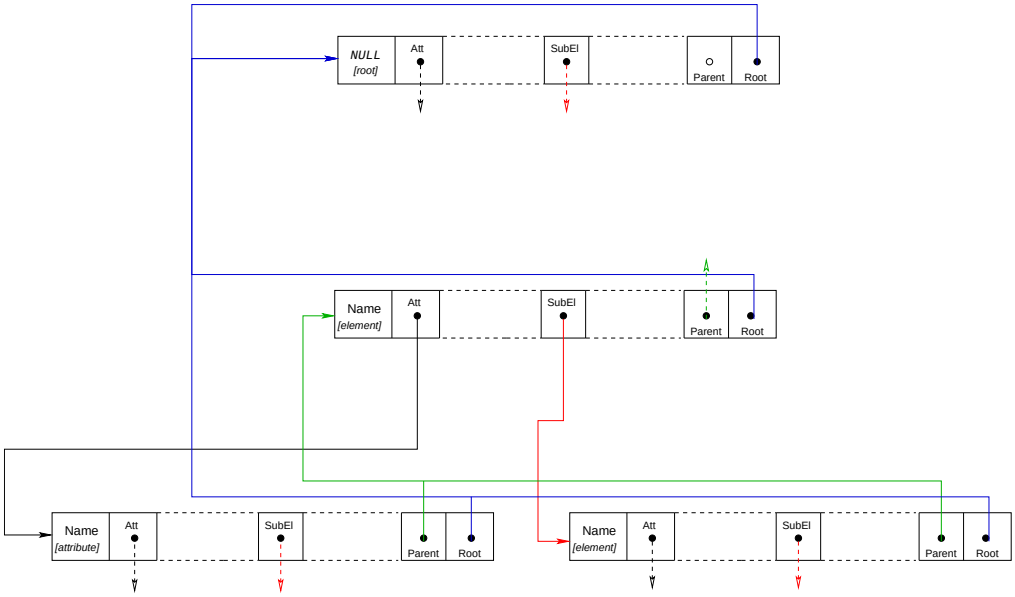
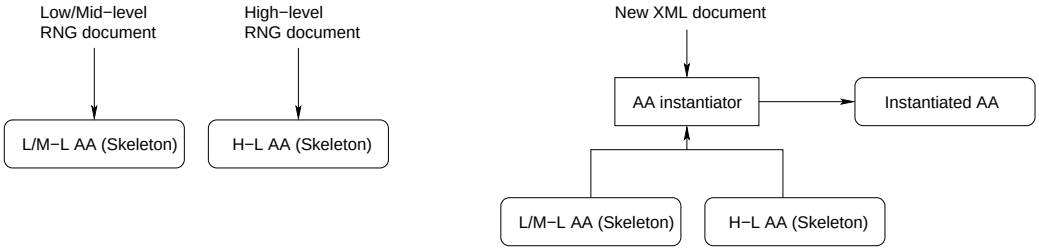


Figure 7.2 • *Internal structure of an autotroph agent*



(a) The skeletons are created at system initialization

(b) The matching skeleton is replicated and instantiated

Figure 7.3 • *Skeleton creation and use*

interact much with their environment, notably during the reification phase. What highly differentiates them from classic structures such as vectors is that they are aware of their own structure and can completely modify and update it.

Even though AAs cannot be considered as pure agents, we showed they represent much more independent and smart structures than traditional object approaches. Furthermore they are very similar entities to those addressed by the KidSim and IBM agents defined in point 1.2.1. Consequently, even if disputable, we classified them as data-oriented agents.

7.1.2 3Store

3Store is the *knowledge base* (KB) which contains all the analysis results retrieved from sensors. The AA storage will happen asynchronously for performance reasons. Consequently, a system failure might result in data loss. Details regarding the KB selection will be thoroughly addressed in section 7.3.

7.1.3 Cache

The cache memory intends to speed up queries. Aware of the huge amount of disk access to perform, keeping as much information as possible in main memory was of the utmost importance.

The cache memory was implemented by means of a red/black tree guaranteeing a query, insertion and deletion time of $\mathcal{O}(log\ n)$ [CLRS02]. Each node in the tree is an AA whose access, comparison and deletion key is its unique id determined by a 64-bit integer.

Given the size and incoming rate of data, the cache size has to be limited to avoid excessive swapping and main memory overflow. In order to meet the imposed real time constraints, the size must be determined by the needs of the high-level analysis. The cache size has thus to be as close as possible to the maximum interval between the latest received file, triggering a high-level analysis, and the oldest previously received file used by the triggered analysis. So, if we assume that the oldest file used is 5 minutes old and that the system receives 25 files per second, the number of agents to keep in cache would be $5 \times 60 \times 25 = 7500$, i.e. a 7500-node red/black tree.

7.2 Knowledge structure management

This section addresses the core of the DW we developed, i.e. the knowledge structure. In the first place the schema of the used RDF graph and the way XML documents are converted into RDF will be presented. In the second place the handling of queries and reification will be described.

7.2.1 RDF graph schema

In order to reduce the amount of stored knowledge and to guarantee a minimal system response time, the graph structure must be flatten to its minimum. Consequently, the possibly deep hierarchies present in XML documents must be crushed. We mentioned in point 7.1.1 that AA skeletons were defined by means of RNG documents. As a result, both low/mid and high-level results are specified by means of RNG documents only known at run-time, consequently ruling out any *a priori* optimization. Moreover the stored knowledge only has to be accessed locally. As a result, URLs used to identify resources can be specifically adapted to our needs.

```
1  <CARETAKER>
2    <Results>
3      <Frame>
4        <Tracked_target>
5          <Object object_id=1>
6            "Luggage"
7          </Object>
8          <Object object_id=2>
9            "Vending machine"
10         </Object>
11       </Tracked_target>
12     </Frame>
13   </Results>
14 </CARETAKER>
```

Figure 7.4 • *Source XML document*

The conversion of XML results into RDF is defined as follows. The enclosed examples are based on the XML document[4] presented in figure 7.4.

1. Create a node having for `urn` the type of low/mid-level analysis, let us say `Object-Recognition`, and append the id of the received file. A particular instance of this resource would look like:

   <urn:ObjectRecognition#11666>

   This node is the document instance node. It is linked to the RNG type node looking like:

   <urn:ist-caretaker.org/#ObjectRecognition>

2. Create nodes corresponding to the instantiated fields of the XML files. The `urn` of a given field will be composed of the full path going from the root to the specified field. In order to guarantee resource uniqueness, the id of the instance will be appended. Numbers will also be added between fields along the path to avoid duplicated field name conflicts. A particular field resource would look like:

   <urn:CARETAKER/0/Results/0/Frame/Tracked_target/0/#id11666_1_Object>

3. Associate the value node to the field node. Two cases can occur:

   (a) the value belongs to the field. In this case the predicate will have a generic label, for instance:

   <urn:xml2rdf/edge/#content>

   (b) the value belongs to an attribute of the field. In this case the predicate will be named by the attribute, for instance:

   <urn:ist-caretaker.org/#object_id>

4. Add a type node to every field node. A particular instance of this resource would look like:

   <urn:xml2rdf/type/#Object>

   We will see in point 7.2.2 what they are used for.

As a result, the depth obtained for any XML input remains constant. A generic RDFS document corresponding to this transformation is defined in figure 7.5.
A subgraph will typically have for triples:

```
1  (1 x) <relaxNG_Type> <hasInstance> <instance_ID>
2
3  (N x) <instance_ID> <instContent> <field_ID>
4        <field_type> <typeInstance> <field_ID>
5        <field_ID> <hasContent> ''content''
```

with a constant depth of four, which sharply reduces possible graph depth. Figure 7.6 illustrates the result of the above delineated conversion process.

[4] It is a very reduced sample of a typical low/mid level analysis result.

```
<rdf:RDF     xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
             xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
             xmlns="urn:xml2rdf/#"
             xml:base="urn:xml2rdf/#">

  <!-- CLASS definitions -->

  <rdfs:Class rdf:ID="relaxNG_Type_Class"/>
  <rdfs:Class rdf:ID="instance_ID_Class"/>
  <rdfs:Class rdf:ID="field_ID_Class"/>
  <rdfs:Class rdf:ID="field_type_Class"/>

  <!-- PROPERTY definitions -->

  <rdf:Property rdf:ID="hasInstance" >
    <rdfs:domain rdf:resource="#relaxNG_Type_Class"/>
    <rdfs:range rdf:resource="#instance_ID_Class"/>
  </rdf:Property>
  <rdf:Property rdf:ID="instContent" >
    <rdfs:domain rdf:resource="#instance_ID_Class"/>
    <rdfs:range rdf:resource="#field_ID_Class"/>
  </rdf:Property>
  <rdf:Property rdf:ID="hasContent" >
    <rdfs:domain rdf:resource="#field_ID_Class"/>
    <rdfs:range rdf:resource="rdfs:Literal"/>
  </rdf:Property>
  <rdf:Property rdf:ID="typeInstance" >
    <rdfs:domain rdf:resource="#field_type_Class"/>
    <rdfs:range rdf:resource="#field_ID_Class"/>
  </rdf:Property>
</rdf:RDF>
```

Figure 7.5 • *Generic RDFS for stored knowledge*

A more relevant example of a video analysis result is available in the accompanying CD under the `XML2RDF` directory. Two RDF documents corresponding to this sample XML are also present. The first one is the result of a naive conversion from the XML while the second has been obtained with our conversion algorihm. The RDF graphs generated by the W3C validator[5] have also been included for illustration purpose. Please refer to the `synopsis.txt` file for more information about file names and contents.

7.2.2 RDF graph handling

Reification

Reification was the main reason for which optimizations were required. Besides being space consuming, deep hierarchies entail much SPARQL queries and recursive browsing. Thanks to the graph schema defined in point 7.2.1, the reification of an AA can be performed with a single SPARQL query:

```
SELECT ?fieldID ?hasContent ?content
WHERE {
              <instance_ID> <instContent> ?fieldID .
              ?fieldID ?hasContent ?content .
       }
```

[5]For further information about the W3C validator please visit `http://www.w3.org/RDF/Validator/`

```
1  <urn:ist-caretaker.org/#ObjectRecognition>
2    <urn:xml2rdf/edge/#hasInstance>
3    <urn:ObjectRecognition#11666>
4  <urn:ObjectRecognition#11666>
5    <urn:xml2rdf/edge/#instContent>
6    <urn:CARETAKER/0/Results/0/Frame/Tracked_target/0/#id11666_0_Object>
7  <urn:ObjectRecognition#11666>
8    <urn:xml2rdf/edge/#instContent>
9    <urn:CARETAKER/0/Results/0/Frame/Tracked_target/0/#id11666_1_Object>
10 <urn:xml2rdf/type/#Object>
11   <urn:xml2rdf/edge/#typeInstance>
12   <urn:CARETAKER/0/Results/0/Frame/Tracked_target/0/#id11666_0_Object>
13 <urn:xml2rdf/type/#Object>
14   <urn:xml2rdf/edge/#typeInstance>
15   <urn:CARETAKER/0/Results/0/Frame/Tracked_target/0/#id11666_1_Object>
16 <urn:CARETAKER/0/Results/0/Frame/Tracked_target/0/#id11666_0_Object>
17   <urn:ist-caretaker.org/#object_id>
18   "1"
19 <urn:CARETAKER/0/Results/0/Frame/Tracked_target/0/#id11666_0_Object>
20   <urn:xml2rdf/edge/#content>
21   "Luggage"
22 <urn:CARETAKER/0/Results/0/Frame/Tracked_target/0/#id11666_1_Object>
23   <urn:ist-caretaker.org/#object_id>
24   "2"
25 <urn:CARETAKER/0/Results/0/Frame/Tracked_target/0/#id11666_1_Object>
26   <urn:xml2rdf/edge/#content>
27   "Vending machine"
```

Figure 7.6 • *XML source converted into RDF triples*

Query

As figure 7.1 shows, `select` operations are both performed on cache and 3Store. Let us precise how they are combined to ensure that every matching reply is retrieved:

1. convert the XML query into an AA;
2. apply it as a mask on every AA present in cache. If the masked AA still shows instantiated elements, it is accepted. Ids of matching AAs are stored in a list l_c;
3. convert the XML query into several SPARQL queries;
4. run queries and consider only results whose ids are not in l_c. Ids of 3Store matchings AAs are stored in a list l_{3S};
5. AAs from l_{3S} are reified, the query is applied as a mask to them and they are exported into XML;
6. AAs from l_c are retrieved from cache and they are exported into XML.

Note that no cache has been implemented for high-level events. As a result, only the SPARQL related part must be considered.

An important thing to note is that to every `<field_ID>` a type node is associated to optimize query results. Indeed, common values such as 1 or 2 abundantly come over in analysis results. As the field <subject> cannot be precised because neither its instance id nor path position is *a priori* known, type nodes were used to restrict query results. Consequently, a field query will typically look like:

```
1  SELECT ?fieldID
2  WHERE {
3                <field_type> <typeInstance> ?fieldID  .
4                ?fieldID <hasContent> "1"  .
5          }
```

In the typical subgraph structure presented hereabove it may seem unnatural to present the triple as:

```
1  <field_type> <typeInstance> <field_ID>
```

rather than as:

```
1  <field_ID> <hasType> <field_type>
```

The reason is that, at reification time, the performed query will additionally return the types of matching data. As a result, the returned amount of data may be doubled, which reduces search performances.

7.3 Triple store selection

The RDF physical storage is a burning question in the world of *Knowledge Base Management Systems* (KBMS). It is profusely addressed in the literature and notably in [Owe05, HG03, TCK05, LH05, DWSK03, BHS03]. Graphs may be stored in an object-oriented database, in a relational database or even in text files. Some use RDFS and/or OWL to define table schemas where others use a single table containing triples. The main issues to solve are query performances, storage space, set up and maintenace costs, robustness, inference and reasoning mechanism specifications, and data structure conversions. These concerns will not be discussed here as they are out of the scope of this document. Nevertheless, in order to vindicate our choice, we present in table 7.1 an overview of the possibly expected features of a KBMS. The remaining of this section will focus on 3Store and present it more thouroughly.

Table 7.1 • *Overview of the main RDF-capable KBMS available*

KBMS	Volume	Performance	RDFS	OWL
3Store[HG03, GH]	✓	✓	✓	
Jena[JEN, CDD+03]			✓	✓
Kowari[KOW, WGA05b]	✓		✓	✓
Redland[Bec01, RED]		✓	*U*	
Sesame[SES]	✓		✓	✓
YARS[YAR]	✓	✓	*U*	*U*

✓: function available ; *U*: unknown availability

The selection of 3Store [HG03] as the KBMS of our system was made for the following reasons:

- the amount of storable triples is very important (several millions);
- the use of *MySQL* allows to benefit from the optimizations brought along by the relational engine;
- the triples storage is mainly based on two tables:

  - *triples*: this table contains the set of RDF triples, which are stored as hashed values;
  - *symbols*: this table notably contains the mappings from triples to the hashed values used in the *triples* table;

  The amount of joints is thus sorely reduced, which guarantees a good response time. Moreover, the use of hash instead of textual elements improves query performances as it is working with numbers rather than arbitrary strings;

- the triples stored in the *triples* table are actually stored as quadruples[6]. Consequently, it is possible to take graph models, also called contexts, into account.

- the inference mechanisms proposed are presented in table 7.2 and are reduced to a very limited set. The non-use of OWL inference capabilities improves system performances. Furthermore, we have seen that the inference performed on low, mid and high-level knowledge is handled by external entities. It would thus be useless to start new analysis based on ontologies within the system. The chosen perspective is thus the distributed reasoning;

Table 7.2 • *3Store inference rules*

Label	Rule	Implication
direct class	xxx rdfs:subClassOf yyy	xxx direct:subClassOf yyy
direct property	xxx rdfs:subPropertyOf yyy	xxx direct:subPropertyOf yyy
rdfs2	aaa rdfs:domain xxx uuu aaa yyy	uuu rdf:type xxx
rdfs3	aaa rdfs:range xxx uuu aaa vvv	vvv rdf:type xxx
rdfs8	uuu rdf:type rdfs:Class	uuu rdfs:subClassOf rdfs:Resource

- the supported query languages are SPARQL and RDQL[7];

- unlike other KBMS, the data access protocol stack is very light. This offers the possibility to tune the low-level access mechanisms for the application domain.

Note that the RDFS will not be explicitly exploited in the TS. We preferred to leave the inference and optimization manipulations to the AAs.

[6] 3Store is, as opposed to what one could think, a *quad store* rather than a TS as stated in [LQS].

[7] The *RDF Data Query Language* (RDQL) [RDQ] is another RDF query language. It has not been discussed because it is now being superseded by SPARQL.

Part III

Synthesis

8 System Evaluation

Evaluate what you want – because what gets measured, gets produced.

James A. Belasco

OVERVIEW

This short chapter will evaluate the agent system which has been developed. In the first place, the system performances will be assessed. In the second place, the system questionable points will be addressed. Finally, perspectives of evolution will be put forward.

8.1 Performance

Tests were performed with sample XML files retrieved from video sequences and are summarized in table 8.1. Simulators were used to send low/mid/high-level analysis and queries. The table lists the results of some of the most recurring operations. The XML test file used here comprises 53 instantiated values. The 3Store knowledge base contained more than 4 000 000 triples, corresponding to approximately 13 minutes of recording at 25 files per second[1].

Table 8.1 • *System performance evaluation*

Operation	Average time (ms)
Reification (from cache)	0
Reification (from 3store)	40
Query (from cache)	3
Query (from 3store)	50
Agent replication	0
Agent initialization	5
Agent → XML	5
Agent → RDF	5
Insertion (in cache)	0
Insertion (in 3store)	70

Unfortunately, real-scale data sets were not available when performing these evaluations. Consequently, and much to our regret, these measures must only be regarded as indicative. Note that with the transformation performed from the initial XML schema directly converted into RDF we obtained a reification time from 3Store of about 1600 ms, i.e. 40 times slower than with our optimization.

8.2 Discussion

This section will pinpoint some of the system disputable aspects. Improvements to some of them will be presented in section 8.3.

Firstly, and maybe unexpectedly, why did we use agents ? Indeed, the system seems closed, the autonomy of agents appears somewhat limited and the proactive behaviours look missing. Remember we said at point 5.1.2 that the designed system was not expected to exploit every possibility of AOP. Moreover, the system complexity and the high degree of concurrency would have been very difficult to manage with a multi-threaded OO approach. The interaction and coordination mechanisms offered by the FIPA highly simplified the development regarding these issues. Furthermore, the agent intrinsic high-level view proposed during the design process eases the component identification and role descriptions. Consequently, given the high degree of coherence and low level of coupling among the entities, resorting to AOP was, in our opinion, the most appropriate choice.

Secondly, the `RSSFlowHandler` agent can be seen as a bottleneck preventing the system from running as fast as it could. The agents interacting with it could be addressed directly. Let us attempt to justify our choice. In the first place, the performed tests showed that the system suffers no slow down due to the centralization of the external data flow handling.

[1] Note that we took here an extreme case as usual low/mid-level analysis results are closer to 10 files per second.

One must not forget that agents run plans, each plan achieving a particular task. The `RSSFlowHandler` must thus be regarded as a dispatcher rather than as flow converter. Its task is to start the plan in charge of some given input. It is the plan that is responsible to handle and address it correctly. Moreover, given that each plan is an independent Java thread, which may be duplicated in case of overload, the system flexibility and reactivity does not suffer from this centralization. In the second place, the use of a single agent interacting with the outside world guarantees more consistency and highly facilitates agent management.

Thirdly, why resorting to RDF while traditional relational databases show excellent performances in handling huge amounts of data ? Exactly, they handle data. The metadata produced by the video analysis is much closer to knowledge than raw data. Furthermore, their complex and undefined structure prevented any *a priori* table schema definition from being built.

Fourthly, the goal, and consequently the use, of the `DataHandler` agent could appear questionable. A valuable answer to this question is actually tricky to formulate. Originally, the goal of this agent was mainly, as presented in previous chapters, to extract additional knowledge from high-level analysis and only merge it with the RDF graph stored in the DW. Once again, the data schema was very loosely defined and prevented thus any real reflexion about their integration. As a result, the `DataHandler` potential is currently highly under-exploited.

Finally, the number of needed conversions is way too high. A query is for example converted six times between sending and reception. The interaction with heterogeneous systems unfortunately conditioned them.

8.3 Perspectives

Figure 8.1 shows a streamlined view of the original system presented in figure 5.5. Figure 8.2 illustrates how the proposed evolutions impact on the original design presented in figure 5.6. The advantages of this approach will be discussed hereafter. They will be followed by some thoughts about the feasibility of such a novel design.

Firstly, far fewer conversions are needed and consequently less work needs to be performed at the demonstrator's level. As a result, data consistency is guaranteed.

Secondly, a much better and complete use of the RDF technology is put forward. As figure 8.3 shows, the different levels of video analysis results could easily fit into the RDF stack. Besides being technically more adapted, it is, in our opinion, closer to what these results intend to model, i.e. knowledge acquired about the perceived environment. Note that we did not transform the XML query directly into SPARQL for two reasons. Initially because the query mechanism offered in our XML queries is more powerful than in an SPARQL query. More than one SPARQL query would thus be needed to retrieve the same results, which increases transmission and thus response time. In the second place, as there is no widely accepted RDF query standard, it appeared wiser to avoid losing generality by imposing a particular language. Alternatively, RDF specific queries could be wrapped in a single RDF document, which will then be analyzed to retrieve and run them.

Thirdly, agents can be used more efficiently. They are indeed more widely distributed in this open environment. They could possibly be endowed with more autonomy and proactivity and thus reflect the agent paradigm more reliably.

Fourthly, the global architecture appears less compartmentalized and heterogeneous than in previous versions.

Finally, the possible introduction of *Video Content Analysis* (VCA) capable agents in the

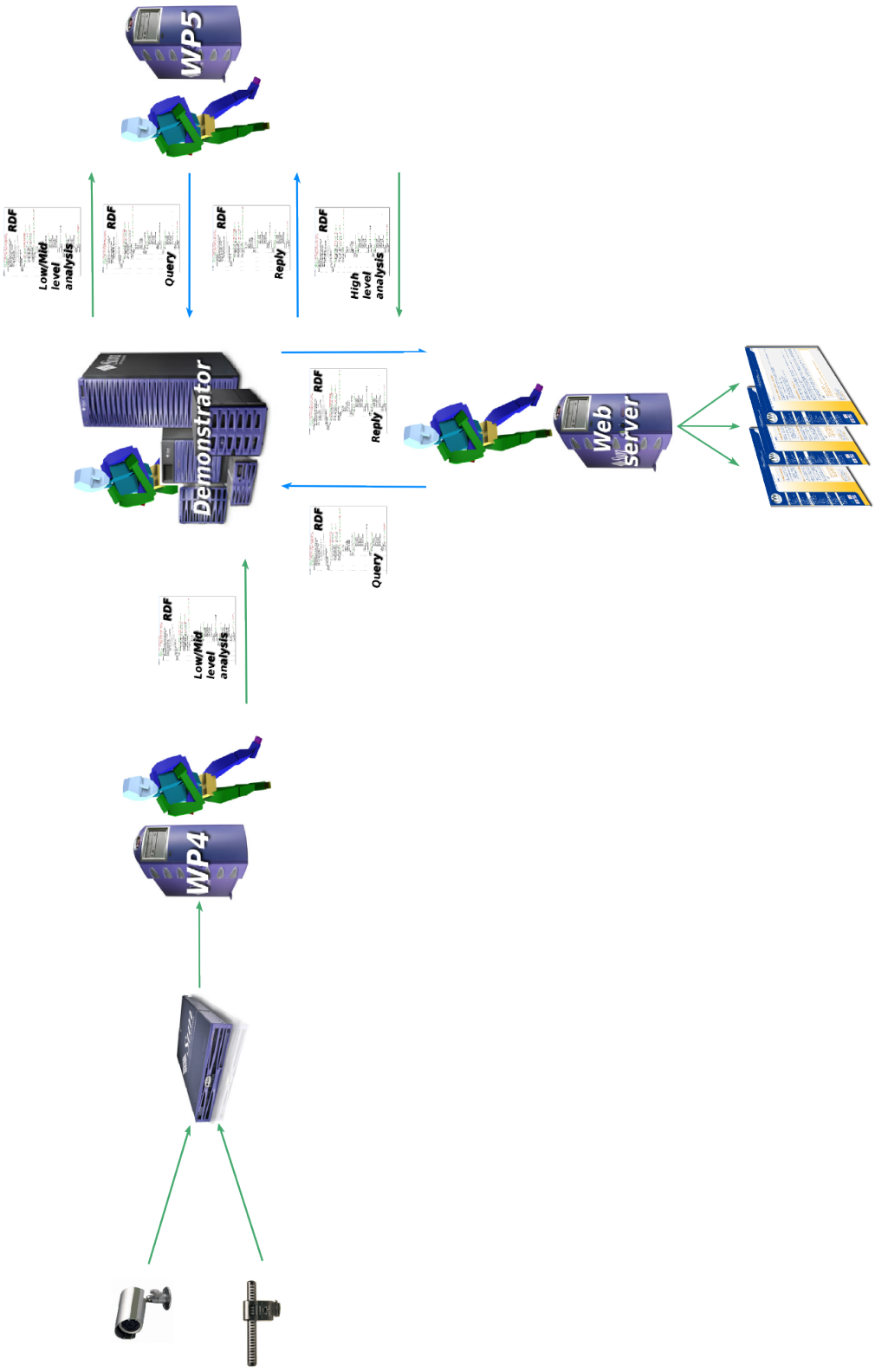


Figure 8.1 • *System framework evolution*

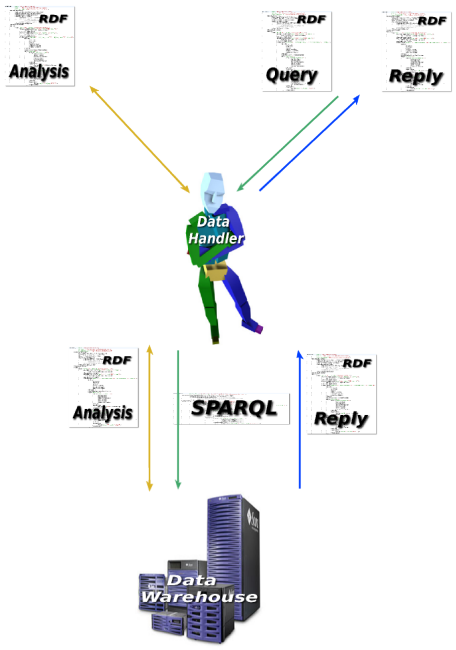


Figure 8.2 • *System abstract architecture evolution*

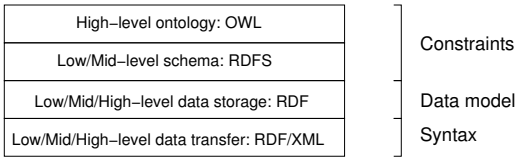


Figure 8.3 • *Video analysis adapted RDF stack*

system will be greatly facilitated as no new specific protocol would be needed. For instance every sensor could be coupled with its VCA agent directly communicating with the system.

Nevertheless, current practices in the domain still prevent this evolution from being possible. The definition of data standards in current video analysis trends is more focused on the content and on the possible information to include in it rather than on the way it will be stored or presented. In other words, the concerns are rather focused on processing than on possible *a posteriori* usage. There are also minor concerns about their use in the SW, but we can definitely not blame them for this.

Furthermore, the enclosure of VCA capable agents directly in the sensors may not always be possible as proprietary material is mostly used. Agents would also prevent the reuse of this component in totally different environments. The standardization of VCA agents is still far from conceivable, which is a major barrier to their uptake.

However, such agents could be of great help in *Closed Circuit TeleVision* (CCTV) where data synchronization is still a major issue. The physical distribution of sensors and their increase make tough data handling from a human perspective. Consequently, the smart and automatic handling of this amount of distributed information assumes a crucial character. Moreover, the current reliability of video analysis algorithms allows one to focus mainly on high-level concerns in a distributed environment.

In order to develop such a distributed video analysis system, several environment elements need to be specified. An ontology has to be defined, scenarios identifiable during the high-level analysis have to be precised and the synchronization and centralization of the results have to be assured. Note that unlike current trends, the proposed approach intends to offer a generic video analysis technology.

The application would include two parameters, a domain ontology and a set of scenarios. The ontology has to be described to identify domain specific features. Higher level processing such as complex event recognition will be achieved by means of automata, each specifying a scenario. The settlement of these parameters would enable the inference engine to determine from received low/mid-level information, currently activated automata states and the ontology the next accepting states. Current states could then be updated accordingly. If new scenarios are identified, they are integrated into the system. Besides determining new states, probabilities could be associated to transitions according to machine learning models such as hidden Markov models. Such an architecture offers an incremental evolution of the system's analysis status. Any *a posteriori* analysis, too consumming in real-time environments, is consequently avoided.

The integration of sub-results into the global system will then be needed. In order to minimize the data to transmit, optimizations should be performed on automata to cut down their size. This would lighten the result transmissions and ease their aggregation. So as to handle this flow, specialized agents could be deployed. A centralized system would be responsible to store the produced knowledge and present it to the end users.

This system is thus independent from video analysis algorithms and its behaviour is guided by the domain ontology and the pre-defined automata.

9 Conclusion

This is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.

Sir Winston Churchill

OVERVIEW

With a view to depict how our work contributed to the video analysis sharing area, our vision of both programming and data management techniques will first be put forward. Future work regarding system uses and extensions will then be addressed.

9.1 Our vision

Michelangelo di Lodovico Buonarroti Simoni (1475-1564) was one of the most talented architects, sculptors and painters of the Italian Renaissance. He painted world known frescos on the ceiling of the Sistine Chapel among which *The Creation of Adam*. The striking element of this master piece is with no contest the Hand of God giving life to Adam or more precisely the slight distance separating God's from Adam's finger. This fresco portrays the intent of the present work.

Agents, the RDF, the SW and system engineering are concepts originating from different domains. Agents, and by extension agent system engineering, could belong to a class regrouping programming techniques. The RDF, and by extension ontology specification languages, could belong to a class regrouping data management techniques. Arguably these two coarse-grained classes are not strictly distinct. Nevertheless they formerly tended to share far fewer elements.

The key point to notice throughout the previous chapters was the increasing convergence between both domains. Agents are entities endowed with reasoning capabilities, mentalistic aspects and knowledge about their environment. This combination makes them very useful in concurrent, distributed and highly-reactive structures. The definition of evolutive ontologies, strongly supported by the SW, promotes knowledge development and sharing. What differentiates the SW approach from any other is the machine-processable dimension. It enables the introduction of knowledge in almost every possible unit formerly handling well formated data such as XML. Moreover, the axiomatization of knowledge structure by means of formal languages implicitly integrates processing into knowledge.

Obviously, these two classes put forward share many things. Nevertheless, a tiny gap between active entities carrying knowledge and processable knowledge still exists. But does this slight distance need to be filled ? Modern science shows a highly specialized and compartmentalized trend in research efforts. Knowledge engineering and software engineering do not depart from this rule. However, recent concerns attempt to break these barriers. The SW is a striking example of this move. The *e-science* grid, discussed notably in [Hen05, DRH04], is a vivid illustration of this new progress toward more integration. Integration intends to offer a less segmented view of science and to promote open interactions. It does not aim at regrouping sciences into one but to enhance interactions and consistency among them.

The proposed video analysis sharing system is also a patent example of this convergence. Highly specialized algorithms used in video analysis are coupled to software agents to form a complex reasoning system. As complexity grows, the knowledge to handle increases as well. Consequently, highly capable frameworks are to be put in place in order to meet the required flexibility and availability.

9.2 Contributions

These illustrations of conceptual distances between research concerns, however, show indubitable relations among them. There is no need of explicit links between them as they arise naturally as specific interactions become mandatory. In the system we developed this key role is played by AAs, defined in point 7.1.1. They are the units connecting intelligent agents to knowledge. The toughness we faced when defining them was intrinsic to their position in the system. They belong to both and neither of these two categories. Linking heterogeneous domains can only be achieved by means of such hybrid constructs. They are the success factors in the creation and maintenance of evolutive systems. Moreover, they foster reusability. Even though being higher-level abstractions, agents and RDF are

as handable as objects and tables. As abstraction increases, lower-level components can be changed or reused without any noticeable alteration of high-level elements. However, from the implementation point of view, major changes on both sides might be required. In our case, AAs can be seen as mid-level abstraction entities. If changes are to be made on the storage engine for instance, only the AAs will have to be modified and the agents handling them will not have to be altered.

This system naming also turns out to be a tricky business. *Knowledge Management System* would be too restrictive as it does not reflect the complexity of the knowledge independent processes. *Complex Event Management System* focuses more on the algorithmic complexity of the work to achieve than on its memorization. *Smart system* appears to be a good tradeoff. It encompasses both intelligence and knowledge management concepts. Furthermore, the connotation brought by the *smart* notion illustrates the link with the mentalistic notions it derives from or entails.

We demonstrated too that the combination of autonomous and data-oriented agents to SW technologies proved extremely powerful in a real-time environment. The natural ability of agents to interact with each other in an asynchronous way entails a highly dynamic behaviour and the flexibility of the knowledge structure proposed by the SW greatly eased the design of an adaptable and flexible system. The association of these components made possible the design of a generic, context-independent and scalable knowledge sharing system particularly suited for video analysis sharing.

9.3 Future work

The processing of video analysis results can be considered here as a simple framework for knowledge providing. In other words, any application producing computation results in XML and requiring an efficient knowledge retrieval mechanism could resort to our system without any architectural modification. Moreover, given its scalability capabilities, multiple low/mid-level, high-level analyzers and web servers can be connected to it without any transformation. Minor changes to the implementation could enable the synchronization among low/mid and high-level analyzers. Possibly candidate domains could be grid computing, stock exchange speculation computing, future selling estimations from buying behaviours, possible path selection from GPS coordinates, ...

The interactions with the Web server could be improved by offering a higher level view of the knowledge. This could be achieved by means of the design of a specific RNG document only gathering information relevant to the users. Information from both low/mid and high-level RNGs should consequently be merged and cleaned from inessential parts. Furthermore agents responsible for the notification of particular high-level events to the Web server should be developed to improve the system reactivity and pro-activity.

In order to face the storage space issue, automatic deletion mechanisms could be set up. One can reasonably assume that very low-level elements will be useless after some minutes or hours of recording. A special agent continuously monitoring and wiping out the DW could then be created.

We did also show in point 8.3 how our approach can be extended to meet distributed video analysis specific needs. The intent of this extension is to promote AI, agents and SW technologies as a framework for high-level knowledge handling and sharing.

Part IV

Appendix

Specifications

It is the mark of an educated mind to rest satisfied with the degree of precision which the nature of the subject admits and not to seek exactness where only an approximation is possible

Aristotle

OVERVIEW

In this chapter, the syntax and the semantics of the exchanged data, queries and masks will be presented.

A.1 RSS

A.1.1 Required fields

The elements of the RSS 2.0 specification presented in table A.1 are the only tolerated ones. The values they must contain are defined in point A.1.2.

Table A.1 • *RSS feeds specification*

XML tag	Value	Cardinality
`<rss version="2.0">`		1-1
`<channel>`		1-1
`<title>`	*title*	1-1
`<link>`	*link*	1-1
`<description>`	*description*	1-1
`<category>`	*category*	1-1
`<item>`		0-*
`<title>`	*id*	0-1
`<description>`	*itemContent*	1-1
`<enclosure>`	*encl*	1-1

A.1.2 Syntax

```
1  <string>          ::= [A-Z|a-z][A-Z|a-z|0-9|_]*
2  <URL>             ::= # go to the URL BNF specification page
3  <long>            ::= [0-9]|[1-9][0-9]*
4  <sep>             ::= "_"
5  <project>         ::= "CARETAKER"
6  <xmlFile>         ::= # go to the XML BNF specification page
7  <title>           ::= <sourceName> <sep> <numberOfItems>
8  <sourceName>      ::= <string>
9  <numberOfItems>   ::= <long>
10 <itemID>          ::= <long>
11 <link>            ::= <URL>
12 <id>              ::= # see corresponding type
13 <description>     ::= # see corresponding type
14 <category>        ::= # see corresponding type
15 <itemContent>     ::= <xmlFile>
16 <encl>            ::= # see corresponding type
```

A.1.3 Semantics

Semantic definitions:

- `<long>` : 64-bit unsigned integer;
- `<title>` : the title of the produced RSS feed;
- `<sourceName>` : the source id;
- `<numberOfItems>` : the number of included items;
- `<link>` : the link to the feed sender;
- `<description>` : the description of the RSS feed's contents;

- `<category>` : the feed category;
- `<id>` : the id of the feed that triggers the high-level analysis (the same `id` must be used in the "reply" of the high-level analysis);
- `<itemContent>` : the XML file. Its schema must have been previously defined with the *Relax NG* simplified grammar. This can be a query, a reply or data coming from low/mid/high-level analysis;
- `<encl>` : the id of the schema used to create the contents.

A.1.4 Field specifications

Below are specified the field values according to their types. Note that each feed is uniquely typed, i.e. every included `<item>` has the same type.

Frame

```
1  <frame>          ::= "Frame"
2  <type>           ::= "ObjectRecognition"
3  <id>             ::= <project> <sep> <itemID>
4  <description>    ::= <project> <sep> <frame>
5  <category>       ::= <frame>
6  <encl>           ::= <type>
```

Query

```
1  <query>          ::= "Query"
2  <type>           ::= "ObjectEvent" | "ObjectRecognition"
3  <description>    ::= <project> <sep> <query>
4  <category>       ::= <query>
5  <encl>           ::= <type>
```

Reply

```
1  <reply>          ::= "QueryReply"
2  <type>           ::= "ObjectEvent" | "ObjectRecognition"
3  <description>    ::= <project> <sep> <reply>
4  <category>       ::= <reply>
5  <encl>           ::= <type>
```

Scenario

```
1  <scenario>       ::= "Scenario"
2  <type>           ::= "ObjectEvent"
3  <id>             ::= <project> <sep> <itemID>
4  <description>    ::= <project> <sep> <scenario>
5  <category>       ::= <scenario>
6  <encl>           ::= <type>
```

A.1.5 Implementation details

Besides data format, an implementation-specific requirement must be fulfilled. Every new feed sent must be preceded by 8 `bytes` containing a `64-bit unsigned integer` representing the length of the file transferred.

A.2 Queries

A.2.1 Format

As above-mentioned, XML queries are wrapped in RSS feeds. The RNG schema used for the query depends on the type of the queried element. If elements corresponding to schema S must be queried, the schema used to build the query will be S.

The query's reply will contain every field present in the query. In other words, a mask will be applied so that only the specified fields will appear in the reply (see point A.3).

In order to filter the replies' contents, conditions can be added to elements and attributes.

More precisely, if the content of element e with father d and grand-father c is of no matter but we still want to know its value, the query will simply contain:

```
<c>
  <d>
    <e>
    </e>
  </d>
</c>
```

or:

```
<c>
  <d>
    <e/>
  </d>
</c>
```

Similarly, if the content of the attribute a is of no matter, the query will simply contain:

```
<c>
  <d>
    <e a="">
    </e>
  </d>
</c>
```

On the other hand, if a condition is to be applied, we would use:

```
<c>
  <d>
    <e>
    " condition "
    </e>
  </d>
</c>
```

or

```
<c>
  <d>
    <e a="condition">
    </e>
  </d>
</c>
```

if we follow the conventions outlined above. Note that there cannot be more than one element for a given name and a given father. So the following patterns are not valid:

```
<c>
  <d>
    <e/>
    <e/>
  </d>
</c>
```

```
<c>
  <d>
    <e a="" a="">
    </e>
  </d>
</c>
```

A.2.2 Syntax

The syntax of the `condition` statement is the following:

```
<value>          ::=     # Java STRING
<condition>      ::=     "=" <value>
                         |       ">=" <value>
                         |       "<=" <value>
                         |       ">" <value>
                         |       "<" <value>
                         |       <condition> "AND" <condition>
                         |       <condition> "OR" <condition>
                         |       "(" <condition> ")"
```

A.2.3 Semantics

The semantics of the `condition` statement is the following:

- `<value>`: the value to analyse. It can be a number, a date or a string depending on the value type.
- `<AND>`: the logical $\wedge$.
- `<OR>`: the logical $\vee$.
- `<()>`: the condition inside the brackets is evaluated independently from the rest of the condition statement. It should be used to guarantee the correct evaluation of `<AND>` and `<OR>`.

A.2.4 Implementation details

- To use < and > in XML files, replace them respectively with `&lt;` and `&gt;` to avoid parsing troubles.
- To perform queries on dates, the type defined in the RNG schema must be the `xsd` type `dateTime`. Note that dates are converted into `long` before being inserted in the KB. When they are reified, they are converted into `dateTime` back. Consequently, AAs contain only `dateTime` formatted elements.
- To perform queries on numbers, the type defined in the RNG schema must be one of the `xsd` type :
  - `float`
  - `double`
  - `int`
  - `long`
- To perform queries on strings, the type defined in the RNG schema does not have to be one of the types stated above.

A.3 Masks

A.3.1 Format

Masks have exactly the same syntax and semantics as queries. Actually, a query's reply is the corresponding AA on which the query's mask, converted into an AA, has been applied.

The goal of masks is to filter the data output when performing XML and RDF exports.

A.3.2 Implementation details

The masking function has not been extensively tested and its definition has not been clearly established. This could result in unexpected behaviours. The requirements of the function should thus be clearly identified before any further development. The issue here is the definition of accepted patterns. We have taken the convention that if at least one child of the current element has an invalid condition, the element and all its children are masked. The element is accepted only if at least one of its children verifies the element present in the mask.

The unmasking function is automatically called by the masking function before applying the mask in order to ensure results consistency. Be careful when using an AA on which a mask has been applied. The unmasking function should be used after the needed process ends.

Index

G

H

I

J

K

M

N

O

P

R

S

T

U

V

W

X

Y

Bibliography

[AB07] Ben Adida and Mark Birbeck. RDFa Primer 1.0. Working Draft http://www.w3.org/TR/xhtml-rdfa-primer/, W3C, March 2007.

[ACK+01] Sofia Alexaki, Vassilis Christophides, Gregory Karvounarakis, Dimitris Plexousakis, and Karsten Tolle. The ics-FORTH RDFSuite: Managing voluminous RDF description bases. In *SemWeb*, 2001.

[Age06] Agent Oriented Software Pty. Ltd. *JACK Intelligent Agents: Agent Manual*, 2006.

[AGGPE05] Renzo Angles, Claudio Gutierrez, Asuncion Gomez-Perez, and Jerome Euzenat. Querying RDF data from a graph database perspective. *The Semantic Web: Research and Applications*, 3532/2005:346–360, 2005.

[ASRW02] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta. Agile software development methods - review and analysis. Technical Report 478, VTT PUBLICATIONS, 2002.

[ATO06] Atom Publishing Format and Protocol (atompub). Technical report, IETF, http://www.ietf.org/html.charters/atompub-charter.html, 2006.

[BCT+06] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, Giovanni Rimassa, and Roland Mungenast. *JADE administrator's guide*. Tilab, 2006.

[Bec01] David Beckett. The design and implementation of the redland rdf application framework. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 449–456, New York, NY, USA, 2001. ACM Press.

[BGG+04] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, May 2004.

[BHS03] Valerie Bonstrom, Annika Hinze, and Heinz Schweppe. Storing rdf as a graph. In *LA-WEB '03: Proceedings of the First Conference on Latin American Web Congress*, page 27, Washington, DC, USA, 2003. IEEE Computer Society.

[BIP91] Michael E. Bratman, David Israel, and Martha Pollack. Plans and resource-bounded practical reasoning. In Robert Cummins and John L. Pollock, editors, *Philosophy and AI: Essays at the Interface*, pages 1–22. The MIT Press, Cambridge, Massachusetts, 1991.

[BL98] Tim Berners-Lee. Why RDF model is different from the XML model. http://www.w3.org/DesignIssues/RDF-XML.html, October 1998.

[BL00] Tim Berners-Lee. Semantic Web. In *XML 2000 Conference*, http://www.w3.org/2000/Talks/1206-xml2k-tbl/Overview.html, 2000. Slides.

[BL06] Tim Berners-Lee. Artificial Intelligence and the Semantic Web. In *AAAI-06 Keynote*. AAAI, July 2006. Slides.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001.

[BLM02] Tim Berners-Lee and Eric Miller. The Semantic Web lifts off. *ERCIM News*, 51, October 2002.

[BM05] Dan Brickley and Libby Miller. FOAF Vocabulary Specification . Technical report, FOAF Project, 2005.

[Bro85] Rodney A. Brooks. A Robust Layered Control System For a Mobile Robot. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1985.

[CCG+02] Giovanni Caire, Wim Coulier, Francisco J. Garijo, Jorge Gomez, Juan Pavon, Francisco Leal, Paulo Chainho, Paul E. Kearney, Jamie Stark, Richard Evans, and Philippe Massonet. Agent Oriented Analysis Using Message/UML. In *AOSE '01: Revised Papers and Invited Contributions from the Second International Workshop on Agent-Oriented Software Engineering II*, pages 119–135, London, UK, 2002. Springer-Verlag.

[CDD+03] J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: Implementing the semantic web recommendations, 2003.

[CFJ+04] Harry Chen, Tim Finin, Anupam Joshi, Lalana Kagal, Filip Perich, and Dipanjan Chakraborty. Intelligent agents meet the semantic web in smart spaces. *IEEE Internet Computing*, 8(6):69–79, 2004.

[CG00] Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theor. Comput. Sci.*, 240(1):177–213, 2000.

[CLRS02] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction a l'agorithmique*. Dunod, second edition, 2002.

[CNvB00] Jaron Collis, Divine Ndumu, and Christopher van Buskirk. The Zeus Technical Manual. Technical report, BT, http://labs.bt.com/projects/agents/zeus/techmanual/TOC.html, 2000.

[CYC] What is Cyc? http://www.cyc.com/cyc/technology/whatiscyc.

[DAMa] *About the DAML Language*. http://www.daml.org/about.html.

[DAMb] *DAML+OIL (March 2001)*. http://www.daml.org/2001/03/daml+oil-index.html.

[DAMc] *DAML+OIL (March 2001) Reference Description*. http://www.w3.org/TR/daml+oil-reference.

[DeL99] S. DeLoach. Multiagent Systems Engineering: A Methodology and Language for Designing Agent Systems. In *Agent Oriented Information Systems*, Seattle, OR, 1999.

[DMK+05] S. Dasiopoulou, Vasileios Mezaris, Ioannis Kompatsiaris, V.-K. Papastathis, and M. G. Strintzis. Knowledge-Assisted Semantic Video Object Detection. *IEEE Transactions on circuits and systems for video technology*, 15:1210–1224, 2005.

[DP00] T. Davenport and L. Prusak. Working knowledge: How organizations manage what they know. *Ubiquity*, 2000.

[DPM+04] S. Dasiopoulou, V. K. Papastathis, V. Mezaris, I. Kompatsiaris, and M. G. Strintzis. An Ontology Framework For Knowledge-Assisted Semantic Video Analysis and Annotation. In *4th International Workshop on Knowledge Markup and Semantic Annotation (SemAnnot 2004) at the 3rd International Semantic Web Conference (ISWC 2004)*, 2004.

[DRH04] David De Roure and James A. Hendler. E-science: The grid and the semantic web. *IEEE Intelligent Systems*, pages 65–71, January/February 2004.

[DW03] K. Dam and M. Winikoff. Comparing Agent-Oriented Methodologies. In *Agent-Oriented Information Systems*, volume 3030/2004, pages 78–93. Springer Berlin / Heidelberg, 2003.

[DWSK03] L. Ding, K. Wilkinson, C. Sayers, and H. Kuno. Application-specific schema design for storing large RDF datasets, 2003.

[Eli34] T.S. Eliot. *The Rock*. Faber & Faber, 1934.

[Fer92] I. A. Ferguson. *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents*. PhD thesis, Cambridge, UK, 1992.

[FG96] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Intelligent Agents III. Agent Theories, Architectures and Languages (ATAL'96)*, volume 1193, Berlin, Germany, 1996. Springer-Verlag.

[FHVB04] Flavius Frasincar, Geert-Jan Houben, Richard Vdovjak, and Peter Barna. RAL: An Algebra for Querying RDF. *World Wide Web*, 7(1):83–109, 2004.

[FIP05] FIPA specifications. Technical report, FIPA, http://www.fipa.org/specifications/index.html, 2005.

[Fis94] Michael Fisher. A Survey of Concurrent METATEM - the Language and its Applications. In *ICTL '94: Proceedings of the First International Conference on Temporal Logic*, pages 480–505, London, UK, 1994. Springer-Verlag.

[FvHH+01] D. Fensel, F. van Harmelen, I. Horrocks, D. McGuinness, and P. Patel-Schneider. OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2), 2001.

[Gar] L. M. Garshol. Living with topic maps and RDF. http://www.ontopia.net/topicmaps/materials/tmrdf.html. ontopia.

[GBC+] B. C. Grau, S. Bechhofer, D. Calvanese, G. De Giacomo, I. Horrocks, C. Lutz, B. Motik, B. Parsia, U. Sattler, and P. F. Patel-Schneider. OWL 1.1 Web Ontology Language. Technical report, http://owl1_1.cs.manchester.ac.uk/.

[GH] N Gibbins and S. Harris. 3store. http://www.aktors.org/technologies/3store/.

[GHM04] Claudio Gutierrez, Carlos Hurtado, and Alberto O. Mendelzon. Foundations of semantic web databases. In *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 95–106, New York, NY, USA, 2004. ACM Press.

[GHMP06] Allen Ginsberg, David Hirtle, Frank McCabe, and Paula-Lavinia Patranjan. Rif use cases and requirements. Technical report, W3C, http://www.w3.org/TR/rif-ucr/, 2006.

[GHS03] Nicholas Gibbins, Stephen Harris, and Nigel Shadbolt. Agent-based semantic web services. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 710–717, New York, NY, USA, 2003. ACM Press.

[GL87] Michael P. Georgeff and Amy L. Lansky. Reactive reasoning and planning. In *AAAI*, pages 677–682, 1987.

[Gla96] N. Glaser. *Contribution Knowledge Modelling in a Multi-Agent Framework (the CoMoMAS Approach)*. PhD thesis, Universite Henri Pointcare, Nancy I, 1996.

[GRD07] Gleaning Resource Descriptions from Dialects of Languages (GRDDL). Working draft, W3C, http://www.w3.org/TR/grddl/, March 2007.

[Gru93] T. R. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.

[HBLM02] James Hendler, Tim Berners-Lee, and Eric Miller. Integrating Applications on the Semantic Web. *Institute of Electrical Engineers of Japan*, 122(10):676–680, October 2002.

[Hen99] James Hendler. Is there an intelligent agent in your future ? *Nature*, March 1999.

[Hen01] J. Hendler. Agents and the Semantic Web. *IEEE Intelligent Systems*, (2), 2001.

[Hen04] Jim Hendler. Frequently Asked Questions on W3C's Web Ontology Language (OWL). Technical report, W3C, http://www.w3.org/2003/08/owlfaq, 2004.

[Hen05] J. Hendler. Science and the Semantic Web. http://www.cs.umd.edu/ hendler/presentations/Harvard-IIC-05.pdf, December 2005. Slides.

[Her07] Ivan Herman. Introduction to the Semantic Web. In *International Conference on Semantic Web & Digital Libraries*, Bangalore, India., February 2007. W3C. Slides.

[Hew77] Carl Hewitt. Viewing Control Structures as Patterns of Passing Messages. *Artif. Intell.*, 8(3):323–364, 1977.

[Hey04] Jonathan Hey. The Data, Information, Knowledge, Wisdom Chain: The Metaphorical link, December 2004.

[HG03] S. Harris and N. Gibbins. 3store: Efficient bulk RDF storage, 2003.

[HG04] Jonathan Hayes and Claudio Gutierrez. Bipartite Graphs as Intermediate Model for RDF. Technical Report TR/DCC-2004-2, Universidad de Chile, 2004.

[HHL99] Jeff Heflin, James Hendler, and Sean Luke. Shoe: A knowledge representation language for internet applications. Technical report, 1999.

[HPS04a] I. Horrocks and P. Patel-Schneider. A proposal for an OWL rules language, 2004.

[HPS04b] Ian Horrocks and Peter F Patel-Schneider. Reducing OWL Entailment to Description Logic Satisfiability. *Journal of Web Semantics*, 1(4), 2004.

[HPSvH02] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. Reviewing the design of DAML+OIL: an ontology language for the semantic web. In *Eighteenth national conference on Artificial intelligence*, pages 792–797, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[HWS05] L. Hollink, M. Worring, and A.Th. Schreiber. Building a visual ontology for video retrieval. In *Proceedings of ACM Multimedia, Singapore*, 2005.

[IGGV96] C. Iglesias, M. Garijo, J. Gonzalez, and J. Velasco. A methodological proposal for multiagent systems development extending CommonKADS. In Proceedings of 10 th KAW, Banoe, Canada, 1996.

[IMP05] Impact implementation overview. Technical report, University Of Maryland, 2005.

[Inm02] W. H. Inmon. *Building the Data Warehouse*. Wiley Computer Publishing, third edition edition, 2002.

[JEN] Jena: A Semantic Web Framework for Java. http://jena.sourceforge.net/.

[JPS02] T. Juan, A. Pearce, and L. Sterling. ROADMAP: Extending the Gaia Methodology for Complex Open Systems. In *Autonomous Agents and Multi-Agent Systems*, 2002.

[JW99] N. R. Jennings and M. Wooldridge. Agent-oriented software engineering. In Francisco J. Garijo and Magnus Boman, editors, *Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World : Multi-Agent System Engineering (MAAMAW-99)*, volume 1647, pages 1–7. Springer-Verlag: Heidelberg, Germany, 30- 2 1999.

[KG99] David Kotz and Robert S. Gray. Mobile agents and the future of the internet. *Operating Systems Review*, 33(3):7–13, 1999.

[KOW] Kowari. http://www.kowari.org/.

[Lab] Multiagent & Cooperative Robotics Lab. The agentTool Project. http://macr.cis.ksu.edu/projects/agentTool/agentool.htm.

[LAD04] Michael Luck, Ronald Ashri, and Mark D'Inverno. *Agent-Based software development*. Artech House Publishers, 2004.

[Lam] Patrick Lambrix. Desciption Logics. http://www.ida.liu.se/labs/iislab/ people/patla/DL/index.html.

[LDBD06] B. Lienard, X. Desurmont, B. Barrie, and J.-F. Delaigle. Real-time high-level video understanding using data warehouse. In N. Kehtarnavaz and P. A. Laplante, editors, *Real-Time Image Processing (SPIE 2006)*, volume 6063, pages 40–53, 2006.

[LH05] B. Liu and B. Hu. An Evaluation of RDF Storage Systems for Large Data Applications. In *2005 International Conference on Semantics, Knowledge and Grid (SKG)*, page 59, Beijing, China, November 2005. IEEE Computer Society.

[LHC+07] B. Lienard, A. Hubaux, C. Carincotte, X. Desurmont, and B. Barrie. On the use of real-time agents in distributed video analysis systems. In *IS&T/SPIE 19th Annual Symposium on Real-Time Image Processing 2007*, San Jose, California, USA, February 2007.

[LQS] Large Quad Stores. http://esw.w3.org/topic/LargeQuadStores.

[Mae95] Pattie Maes. Artificial life meets entertainment: lifelike autonomous agents. *Commun. ACM*, 38(11):108–114, 1995.

[MRM02] Luis Menezes, Geber Ramalho, and Hermano Moura. Modular definition of agent-oriented languages using action semantics. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 948–949, New York, NY, USA, 2002. ACM Press.

[MSZ01] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic Web Services. *IEEE Intelligent Systems*, pages 46–53, March/April 2001.

[NRST+06] Milind Naphade, John R. Smith, Jelena Tesic, Shih-Fu Chang, Winston Hsu, Lyndon Kennedy, Alexander Hauptmann, and Jon Curtis. Large-scale concept ontology for multimedia. *IEEE Multimedia Magazine*, 13(3), July-September 2006.

[OIL] *Description of OIL*. http://www.ontoknowledge.org/oil/.

[Ont] Ontolingua. http://www.ksl.stanford.edu/software/ ontolingua/.

[OPB00] J. Odell, H. Parunak, and B. Bauer. Extending UML for Agents, 2000.

[Ope] OpenCyc, http://www.cyc.com/doc/handbook/oe/oe-handbook-toc-opencyc.html. *Ontological Engineer's Handbook*.

[Owe05] Alisdair Owens. Semantic Storage: Overview and Assessment, 2005.

[OWL04a] OWL Web Ontology Language Overview. Technical report, W3C, http://www.w3.org/TR/owl-features/, 2004.

[OWL04b] OWL Web Ontology Language Reference. Technical report, W3C, http://www.w3.org/TR/owl-ref/, 2004.

[PLJ05] Milenko Petrovic, Haifeng Liu, and Hans-Arno Jacobsen. G-topss: fast filtering of graph-based metadata. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 539–547, New York, NY, USA, 2005. ACM Press.

[Pol04] Jeff Pollock. Semantic Technologies Seminar: Using the W3C Standard OWL for Semantic Interoperability. Network Inference, April 2004. Slides.

[PSG06] P.F. Patel-Schneider and B. C. Grau. OWL 1.1 Web Ontology Language Overview. Technical report, http://owl1_1.cs.manchester.ac.uk/overview.html, 2006.

[PW02] L. Padgham and M. Winikoff. Prometheus: A methodology for developing intelligent agents, 2002.

[RC07] Mary Rundle and Chris Conley. Ethical Implications of Emerging Technologies: A Survey. IFAP, Communication and Information Sector, UNESCO, 2007.

[RDF04] Resource Description Framework (RDF). Technical report, W3C, http://www.w3.org/RDF/, 2004.

[RDQ] RDQL - A Query Language for RDF. http://www.w3.org/Submission/RDQL/.

[RED] Redland. http://librdf.org/.

[RK86] S Rosenschein and L Kaelbling. The synthesis of digital machines with provable epistemic properties. In *Proceedings of the 1986 Conference on Theoretical aspects of reasoning about knowledge*, pages 83–98, San Francisco, CA, USA, 1986. Morgan Kaufmann Publishers Inc.

[RN95] Stuart Russel and Peter Norvig. *Artificial Intelligence: a Modern Approac.* Prentice Hall; 1st edition, 1995. p.33.

[RNG03] RELAX NG home page. Technical report, Oasis, http://relaxng.org/, September 2003.

[RSS] *RSS specifications: everything you need to know about rss.* http://www.rss-specifications.com/.

[SBPL04] J. Sudeikat, L. Braubach, A. Pokahr, and W. Lamersdorf. Evaluation of Agent-Oriented Software Methodologies - Examination of the Gap Between Modeling and Platform. In P. Giorgini, J. P. Muller, and J. Odell, editors, *Agent-Oriented Software Engineering V, Fifth International Workshop AOSE*, pages 126–141, 2004.

[SCS94] David Canfield Smith, Allen Cypher, and Jim Spohrer. Kidsim: programming agents without a programming language. *Commun. ACM*, 37(7):54–67, 1994.

[SDWW01] A. Th. (Guus) Schreiber, Barbara Dubbeldam, Jan Wielemaker, and Bob Wielinga. Ontology-based photo annotation. *IEEE Intelligent Systems*, 16(3):66–74, 2001.

[SES] Sesame. http://www.openrdf.org/.

[Sha06] Paul Shabajee. Informed Consent on the Semantic Web - Issues for Interaction and Interface Designers. In *SWUI 2006, The 3rd International Semantic Web User Interaction Workshop*, Athens, Georgia, USA, November 2006.

[SHBL06] Nigel Shadbolt, Wendy Hall, and Tim Berners-Lee. The Semantic Web Revisited. *IEEE Intelligent Systems*, pages 96–101, May/June 2006.

[SHL+05] Chiao-Fe Shu, A. Hampapur, M. Lu, L. Brown, J. Connell, A. Senior, and Yingli Tian. IBM smart surveillance system (S3): a open and extensible framework for event based surveillance. In *IEEE Conference on Advanced Video and Signal Based Surveillance*, pages 318–323, 2005.

[Sho93] Yoav Shoham. Agent-oriented programming. *Artif. Intell.*, 60(1):51–92, 1993.

[Smi80] Reid G. Smith. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Trans. Computers*, 29(12):1104–1113, 1980.

[Smi03a] Barry Smith. *Blackwell Guide to the Philosophy of Computing and Information*, chapter Ontology, pages 155–166. Oxford, 2003.

[Smi03b] Michael K. Smith. Web ontology issue status. Technical report, W3C, http://www.w3.org/2001/sw/WebOnt/webont-issues.html, 2003.

[SOA03] SOAP. Technical report, W3C XML Protocol Working Group, http://www.w3.org/TR/soap/, 2003.

[SPA06] SPARQL Query Language for RDF. Working draft, W3C, http://www.w3.org/TR/rdf-sparql-query/, October 2006.

[SS03] A. Sturm and O. Shehory. A framework for evaluating agent-oriented methodologies, 2003.

[Staa] Stanford KSL Network Services, http://www-ksl-svc.stanford.edu:5915/doc/frame-editor/guided-tour/index.html. *A Guided Tour to Developing Ontologies Using Ontolingua.*

[Stab] Stanford Logic Group, http://logic.stanford.edu/kif/kif.html. *Knowledge Interchange Format (KIF).*

[SWdH+94] Guus Schreiber, Bob Wielinga, Robert de Hoog, Hans Akkermans, and Walter Van de Velde. CommonKADS: A Comprehensive Methodology for KBS Development. *IEEE Expert: Intelligent Systems and Their Applications*, 09(6):28–37, 1994.

[Syc98] K. Sycara. Multiagent systems. *AI magazine*, 19(2):79–92, Summer 1998.

[Syc02] Katia Sycara. RESTINA AFC - Developer's Guide. Technical report, Intelligent Software Agents Lab, 2002.

[Tau06] J. Tauberer. What is RDF. *xml.com*, July 2006. Updated and republished.

[TCK05] Yannis Theoharis, Vassilis Christophides, and Gregory Karvounarakis. Benchmarking database representations of rdf/s stores. In *International Semantic Web Conference*, pages 685–701, 2005.

[Tho93] Sarah Rebecca Thomas. *PLACA, an agent oriented programming language*. PhD thesis, Stanford, CA, USA, 1993.

[Tim01] Bray Tim. What is RDF. *xml.com*, January 2001.

[Tur50] Alan M. Turing. Computing machinery and intelligence. *MIND*, 49:433–460, 1950.

[Vot04] Danna Voth. Biotracking gives back to nature. *IEEE Intelligent Systems*, pages 6–7, January/Ferbruary 2004.

[W3C05] W3C. How people with disabilities use the web. http://www.w3.org/WAI/EO/Drafts/PWD-Use-Web/Overview.html, May 2005.

[WD00] Mark F. Wood and Scott DeLoach. An Overview of the Multiagent Systems Engineering Methodology. In *AOSE*, pages 207–222, 2000.

[Wei66] Joseph Weizenbaum. Eliza: A computer program for the study of natural language communication between man and machine. *Communciations of the ACM*, 9(1), 1966.

[WFCF88] M. T. Weaver, R. K. France, Q. Chen, and E. A. Fox. A frame-based language in information retrieval. Technical report, Blacksburg, VA, USA, 1988.

[WGA05a] Xiaoshu Wang, Robert Gorlitsky, and Jonas Almeida. From XML to RDF: how semantic web technologies will change the design of 'omic' standards. *Nature biotechnology*, 23(9):1099–1103, September 2005.

[WGA05b] David Wood, Paul Gearon, and Tom Adams. Kowari: A platform for semantic web storage and analysis. In *Proceedings of xtech*, 2005.

[Wid95] Jennifer Widom. Research problems in data warehousing. In *IKM '95: Proceedings of the fourth international conference on Information and knowledge management*, pages 25–30, New York, NY, USA, November 1995. ACM Press.

[Win05] M. Winikoff. Towards Making Agent UML Practical: A Textual Notation and a Tool. In *QSIC '05: Proceedings of the Fifth International Conference on Quality Software*, pages 401–412, Washington, DC, USA, 2005. IEEE Computer Society.

[WJ95a] Michael Wooldridge and Nicholas R. Jennings. Agent theories, architectures, and languages: a survey. In *ECAI-94: Proceedings of the workshop on agent theories, architectures, and languages on Intelligent agents*, pages 1–39, New York, NY, USA, 1995. Springer-Verlag New York, Inc.

[WJ95b] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

[WJK00] Michfael Wooldridge, Nicholas R. Jennings, and David Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[WZGP04] Xiao Hang Wang, Da Qing Zhang, Tao Gu, and Hung Keng Pung. Ontology Based Context Modeling and Reasoning using OWL, 2004.

[XML] Extensible Markup Language (XML). Technical report, W3C, http://www.w3.org/XML/.

[YAR] YARS. http://sw.deri.org/2004/06/yars/.

[Yu97] Eric S. K. Yu. Towards modeling and reasoning support for early-phase requirements engineering. In *RE '97: Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)*, page 226, Washington, DC, USA, 1997. IEEE Computer Society.

[ZGMHW95] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View maintenance in a warehousing environment. pages 316–327, 1995.