

# AMIRA

## A State of the Art Analysis of Workflow Modelling and Agent-based Information-Systems

Rouven Thimm, Thomas Will

University of Trier  
Department of Business Information Systems II  
54286 Trier, Germany  
webmaster@{thimmsd.de | cybercosmos.net}

**Abstract.** In recent years the requirements for business information systems have changed. Knowledge- and process-management demand for flexible, dynamic and distributed software environments including the ability to work across the organization's boundaries. With traditional software not up to these challenges new concepts have been developed. This paper reviews the state of the art in workflow modelling, agent technology and agent-based workflow systems. It also pays tribute to the requirements of collaboration and proactivity.

### 1. Introduction

Over the last years there has been a lot of change in the way companies do business. The traditional hierarchical structures are disappearing and being replaced by modern process- or team-organizations. While at the end of the 1990s most companies considered information and information systems to be their most valuable assets, the vast amounts of information accessible today have changed this. People are overwhelmed by the information available and often unable to find exactly what is relevant. Today, instead of focusing on information, the main field of interest is knowledge – To know what the organization really knows. In an age of mergers and joint ventures the combination of business process to one cross-organizational process are of utmost importance.

Traditional software concepts cannot meet these requirements. Modern architectures must be able to communicate across the organization's boundaries, collect information from various sources, support collaboration of humans or software and work as autonomously as possible (there is no time to monitor every single step).

For several years researchers have been working on software concepts that resemble human capabilities. The result is known as agent-theory today. What started out as a hype is slowly beginning to be subject to rational consideration, where and how to model agent-systems. Not every software has to be built on agents, yet there are several appliances where they present just the right capabilities.

With the realization concepts nearly solved the question remains how to model workflows and describe processes that were never meant to be described. Strangely a rather old (in terms of IT) technique, namely Petri nets have proven to be appropriate for workflow modelling, while at the same time new concepts like UML or XML-based semantics are being developed which allow automatic processing of process definitions.

With new projects constantly emerging there is a certain need to present an overview of the available techniques, the problems remaining and important aspects to consider prior to development of new software systems. This paper attempts to present the current state of the art of agent-technology, workflow modelling techniques and the architectural alternatives of agent-based workflow systems. It describes basic concepts as well as concrete architectures and their characteristics.

The rest of this paper is organized as follows: Chapter 2 presents different types of workflows, from traditional fixed to modern flexible concepts. The following chapter deals with representation and modelling using e.g. Petri nets or UML. In chapter 4 the paper describes agent systems, organizational models, communication techniques and discusses

the possible appliances of agent technology to modern software systems. Chapter 5 introduces different agent-based workflow architectures, including performance aspects and details to consider before implementing one or the other. Chapter 6 describes aspects of collaboration and proactivity including their advantages and problems.

## 2. Workflow Modelling

*Workflows* are maps of business processes. They consist of *tasks* and their relations, criteria to indicate start and termination of the processes, and information about the individual task [WFMC99]. Not every business process is a workflow process. A *workflow process* is characterized by three attributes [Aal98]:

- The process is **case driven**.
- The process itself is **essential**.
- The process can explicitly be **defined**.

Nevertheless a workflow is an aggregation of tasks with dependencies defined among them. [AtHu00]. The Workflow Management Coalition (WfMC) uses activity as a synonym for task and defines this as follows: “A description of a piece of work that forms one logical step within a process. An activity may be a manual activity, which does not support computer automation, or a workflow (automated) activity. A workflow *activity* requires human and/or machine resources(s) to support process execution; where human resource is required an activity is allocated to a workflow participant” [WFMC99]. This paper differs from the WfMC-definition in the way that an activity additionally can be an aggregation of atomic activities. From now on an activity will be referred to as task to emphasise this. An aggregation of tasks is subjected to several rules. The three most important are [2002\_J\_IS.pdf]:

- **Membership** (aggregated task can be viewed as a set of activities of a base process)
- **Atomicity** (aggregated task is an atomic unit of processing)
- **Order preservation** (aggregated task preserve the original ordering relations of the base process).

The different tasks are done by one or more actors [EsWi03]. Every actor (from now on called “*agent*”) fulfils a *role* which is defined as a generalized description of the agent’s responsibility for a task [Wan99]. Chapter 4 of this paper will dwell on this topic. The general workflow reference model with its major components and interfaces will not be dealt with in detail here. Details can be found in the official specification published by the WfMC [WFMC96].

The trade press offers three workflow characterizations: *ad-hoc*, *administrative* and *production*. An ad-hoc workflow displays office processes which typically involve human coordination, collaboration, or co-decision. This leads to the problem that task ordering and coordination are not automated and have to be managed by humans instead. The tasks ordering and coordination are accomplished while the workflow is performed. The second type, the administrative workflow, can be characterized as a collection of repetitive, predictable processes with simple task coordination rules. The advantage of this kind of workflows is that ordering and coordination can be automated. Productive workflows consist, like the administrative workflow, of repetitive and predictable processes. In contrast to the administrative workflows, production workflows include a complex information process involving access to multiple information systems. Along the lines of administrative workflows, production workflow can likewise be automated [GHS95]. Due to

the growing importance of knowledge-intensive *weakly-structured* workflows chapter 2.2 will have a closer look at ad-hoc workflows.

Regardless of the workflow's kind in this paper will use the term workflow in two ways. On the one hand as above-mentioned, a workflow is an aggregation of tasks. On the other hand, a workflow can be an aggregation of workflows. The second meaning will be explained in chapter 2.3, where the concept of hierarchical workflows' insertion into a meta-workflow will be described in detail.

A workflow needs several directives in order to fulfil its job. This means: necessary *business rules* are mapped to workflow rules. A business rule can be described best with the following three predications: A business rule is [TaWa01]

- a **statement** about how the business is done,
- a **law** or **custom** that guides an actor's behaviour or actions connected to the organization,
- a **declaration of policy** or condition that must be satisfied.

A workflow has to deliver the right data to that person with the right tool at the right time. The workflow management supports this. Its general purpose is to make sure that the adequate activities are executed by the right person at the right time. [Nr18.pd][Mau01]. These requirements lead to the development of a workflow management system that defines, creates and manages the execution of workflows by using software, running on one or more workflow engines, that are able to interpret the process definition, interact with workflow participants and, where required, invoke the use of external IT tools and applications. Independent of the number of workflow engines number it is indispensable that workflow management systems have the ability to support multiple workflow instances simultaneously. These instances can be seen conterminously with actual business processes [WfMC99].

## 2.1. Workflow concepts

Prior to describing workflow concepts it is necessary to list the general requirements to workflows/workflow concepts in detail. In order to apprehend problems and conditions of workflow modelling, the complexity of office tasks has to be understood. As a consequence thereof several requirements for handling office tasks appear. Their accomplishment requires creativity, initiative and cooperation between the participants. Although it is no problem for human actors to meet these criteria, a software system would have difficulties to do this without a detailed description of the context. Up to now there is no satisfactory approach to describe context or even a task itself good enough for software to precisely understand it. Due to this fact the only thing that can be done is to build systems that help the user to define their goals and their strategies.

Actual workflow management systems rarely provide the flexibility needed. The main requirements can be categorized as follows [LaHa98]:

- **Cooperation support:** Dynamic cooperation between participants must be supported.
- **Tasks-flow management:** Often task patterns cannot be completely determined in advance and require adaptation during runtime.
- **Temporal organization:** Tasks have a high duration and need temporal organization and scheduling.

Based on these requirements, this kind of tasks can be combined under the group of *ad-hoc* workflows.

In literature several concepts were developed for workflow management systems to handle these workflows.

**The Black Boxing Concept.** The black boxing concept is one possibility to solve the dynamic problem. Complex parts of the process are mapped to *black box* sections, which may be defined during their activation and which are unknown for the global workflow. The meta-workflow has to “trust” these black boxes to do their work, whenever they are initialised. Obviously a *hierarchical* structure with at least two levels is embedded in this concept [ABDEHH01]. A survey of hierarchical workflow concepts, problems and requirements will be given in chapter 2.3.

A more specific approach is the **Activity Template Concept**. It is based on the idea that several sets of activity *templates* were created to arrange different activities in a workflow instance. Each template consists of three *attributes*: Input attributes, which list the needed material, assignment attributes, that specify the human responsibilities and time attributes, which define the minimal/maximal duration, the earliest/latest starting time and the warm-up time of the actual task. The latter plays a special role in time critical situations (e.g. fire alert respectively fire extinguishing). An actor should not be surprised by the next activity, contra wise he should know about the next activity on the work list giving him time to prepare himself and his instruments for the next work (warm-up time). A complete planning of a critical situation is not workable. For this reason it is of prime importance to allow dynamic setting and updating of the activities’ attributes. Furthermore the dynamic insertion and deletion of an activity, as well as the dynamic management of work lists is of vital relevance [BKK04].

A third concept refers to a more technical aspect. The **Triggering Concept** alludes to the workflow’s weakest point: The human actor. The workflow management system cannot force an actor to execute a task or to respond once he is finished. This concept tries to counter this problem. Evidently there is a difference between the system enabling a task and requesting a user to execute it. After enabling a task, the actor may execute it, but he could be impeded as well. In an attempt to enable the workflow system to control the process the triggering concept is introduced. A *trigger* is an external condition that leads to the execution of an enabled task. In trigger-controlled workflow systems every user receives an in-basket which holds all tasks a user may work on as of this moment. Whenever the user selects one of these tasks the corresponding trigger is fired, telling the system that he has (re-)started working. It should be noted, that not every task will be by an actor (user).

There are three other types of tasks/triggers [Aal98]:

- **Automatic:** At the moment a task is enabled it will be triggered.
- **Message:** A task will be triggered by an external event (e.g. a message).
- **Time:** A task will be executed by a clock.

## 2.2. Knowledge-intensive weakly-structured Workflows

In the introduction of chapter 2 the differences between several workflow kinds were pointed out. This section will examine the agile ad-hoc workflows in more detail. The main characteristic of ad-hoc workflows is that there is no predefined workflow model for business processes therefore requiring higher flexibility, a better accomplishment with dynamic changes in business rules, implementations, locations and service definition. This

means that ad-hoc workflows are not created a priori, but while they are instantiated and being executed.

For developing efficient ad-hoc workflow concepts it is indispensable to understand the reasons for this ad-hoc behaviour. As implied by this chapter's title, the two main characteristics are knowledge-intensive and weakly-structured. They refer to the attributes of business processes this kind of workflow is designed to map. Knowledge-intensive describes a complex process with many different document-centred activities and few central decisions of human actors. These decisions require comprehensive knowledge about the actual problem, past similar cases etc. The actors need specific knowledge about the activity and the context to in order accomplish it. Weakly-structured on the other hand means that in this kind of process many different actors of several departments with different roles at different locations interact with each other. It is impossible to prescribe the complete workflow with so many unknown variables. Hence this workflow is an unsteady construct which undergoes frequent changes. Complex and informal business processes can be mapped to several black boxes in the workflow, and/or the process can change during its activation [MeHe00] [PaMe03].

### 2.3. Hierarchical Inter-Organisational Workflows

Where is the coherency of hierarchical and *inter-organisational* workflows? An inter-organisational workflow is principally concerned with interoperability between two or more self-contained organisations/actors with *private workflows* and their cooperation in a *public workflow*. Hierarchical workflows on the other hand describe a subordination of workflows to a *meta workflow*. Comparing the meta workflow to the public component and the sub workflows to the private part of the inter-organisation's workflow, a match can be found. But it is superficial to think that there are no differences. The public workflow is not necessarily a meta workflow in a hierarchical sense. It is also possible that a private workflow more or less rules the scenario. For hierarchical workflows the meta workflow determines the general proceeding. Moreover the hierarchical workflow concept does not support privacy aspects whereas the inter-organisational does. So a hierarchical inter-organisational workflow is a combination of both. It contains the hierarchy with a meta workflow defining the main route, but consisting of many sub workflows which are doing their work independently and interacting with the meta workflow in an inter-organisational way.

Several forms of inter-operability are known in literature [Aal99]:

- **capacity sharing:** Accomplishing the workflow under control of one workflow manager. (This belongs to the workflow hierarchy concept, where the meta workflow would be executed by a central workflow engine)
- **chained execution:** Splitting the workflow into a number of disjunctive sub workflows which are executed by different (business) partners in a sequential order. (This approach rather not appertains to a hierarchical concept, because the workflow control is shared.)
- **subcontracting:** In this way of interoperating one actor has got control of the workflow. He is the top-level actor and distributes sub workflows to other subordinated actors. Thus a tree-like hierarchy is generated.
- **Case transfer:** Each actor has a copy of the workflow process description. Cases can be transferred to balance the workload, because tasks may not be implemented at all locations. It is important to consider that at any time, each case exists in exactly one place. If this form uses a central balance manager, a hierarchy is created.

- Nearly the same approach is the **extended case**. It differs from case transfer in the way that here it is possible to diversify the same process definitions at a specific location with additional tasks. This attempt envisages an inheritance concept that will be addressed in the next subsection.
- The last form is called **loosely coupled**. This approach is nearly equivalent to the black box attempt. A process is cut into disjunctive pieces which may be activated in parallel. All sub processes are local and only known to the executing actor. The only public component is the protocol which is used to communicate. This approach rather improper for a hierarchical concept, because a central superordinated instance is missing.

Regardless of the way of inter-operation a few transport policies for carrying cases from one actor to another have been specified. They are based on following restrictions [Aal99]:

- **Minimize the number of transfers**, because this takes time and busies the network
- **Balance the load over actors**. Especially in (time) critical situations it is important not to overload an actor swamping him with the demands.
- **Minimize the throughput time**. The total flow time of cases should be as small as possible, because more traffic causes more network load.
- **Minimizing costs** means that tasks should be executed at the location where processing costs are minimal. In this context cost has a twofold meaning. One the one hand costs in the financial sense, on the other hand costs in a security sense (minimal risk for the actors' life).

Prior to presenting the *transfer policies*, it is necessary to explain the classifications of possible *states*. The four states in a hierarchical order are **active** (a task is being executed), **ready** (at least one task is enabled and is not waiting for an external trigger), **waiting** (at least one task is enabled but all enabled tasks are waiting for an external trigger) and **blocked** (no tasks are enabled).

The above-mentioned transfer policies cut into the following six different forms [Aal99]:

- The **Persistent transfer policy** states that a case will be transferred, if and only if it is in a waiting or blocked state and a transfer leads to a ready state at another location. This policy climbs up the hierarchy.
- In the **strongly persistent transfer policy**, a case will be transferred, if and only if, it is blocked and a transfer leads to a ready or waiting state at another location. This policy climbs up the hierarchy as well.
- In a **time-out transfer policy**, a case will be transferred, if and only if, it has not been active for a specified time and a transfer leads to a ready or a waiting state at another location. If the case was blocked this policy climbs up the hierarchy otherwise it proceeds along the hierarchy (or even descends).
- The **load-balancing transfer policy** states that cases which are ready, waiting, or blocked are transferred to another location if they are not-blocked at the new location and the resulting workload at the new location is smaller. This policy does not belong to a hierarchical transfer. The performance aspect becomes more important.
- In a **cost-driven transfer policy**, cases are transferred to the location where the processing costs for the next task are minimal. This policy does not belong to a hierarchical transfer as well. Here the cost aspect is the relevant item.
- An **Intelligent transfer policy** uses a mixture of the above strategies and also anticipates future developments in their calculations (e.g., the workload in the next period).

Inter-organisational workflows in cooperation with knowledge-intensive weakly-structured processes cause several problems which may be solved by hierarchy. The four main challenges are [Aal03]:

- The **dynamic change problem** occurs if changes were done while the workflow is “running”. Individual changes only affect one single case or a small number of cases, whereas structural changes influence all new cases. Three different ways are possible to handle this problem. The first solution is a restart. This affects all running cases. They have to be rolled back and restarted at the beginning of a new process. The second method is to transfer the case to a new project. The last approach is to proceed. The running cases are not affected by the change, multiple version of the process occur for a (short) period.
- The **management information problem** belongs to the last approach of the dynamic change problem. Multiple variants of one process have to be managed. The solution is a kind of hierarchy, called *inheritance*. This concept demands for an adequate *minimal workflow* so that each variant that is used can be mapped as a subclass or superclass of this workflow.
- The **coordination problem** refers to arrangement problems of two (or more) actors who are involved in one process. Each of these actors is responsible for a part of the workflow. In this way tasks with only local relevance can be added without informing the other actors. This can cause deadlocks, livelocks and other anomalies. The solution is the above-mentioned private/public workflow strategy. It takes four steps to create a workable concept: First, a public workflow has to be developed, second this workflow has to be trenched and distributed to the actors, third a private workflow for every actor has to be created and finally the modification of every actor’s workflow has to be inheritance-preserving.
- The last problem is how to **measure the difference between to processes**. (Automatically) Determining the differences between two processes is necessary to minimize the costs of a process however this causes two problems. Prior to merging both processes it is important to realise where both processes comply. The Greatest Common Divisor (GCD) is a possibility to achieve this aim. Only tasks that are in both processes are absorbed in the GCD. The closer a definition of the GCD is the lesser the number of workflows that come out, and vice versa. So the difficulty is to choose the right ontology for a good outcome.

The question remains why one should use hierarchies. It should be noted that hierarchy can be seen from two perspectives. First of all it addresses the concept of inheritance known from object oriented software development. A workflow that is below a second one in the hierarchy has the same basic tasks and some additional or modified parts. The second form of hierarchy refers to the organizational concept. A workflow consists of a set of tasks. A task itself may again be a complex object (namely a workflow of its own). The complete workflow may be seen as a hierarchy, in contrast to the other form mentioned above there is no inheritance and similarities between workflow and sub-workflow may not exist.

Basically this concept has two advantages. First of all it allows for easier workflow modelling, editing and execution. Parts of the workflow can be viewed as a black box by designer and workflow engine. They are simply passed on to the responsible instance. Moreover this black-box concept allows privacy which is especially useful in cross-organizational workflows. The second advantage is the inheritance which allows reusing of workflow components and central editing.

#### 2.4. Data Auditing (Event-History)

Data Auditing is an essential appliance to increase the process' performance and minimize the costs by reducing the throughput time and optimising the workflow. Obviously the necessity of process analysis is given. The proceeding in process analysis is to identify bottlenecks in a workflow and determine how to redesign as many tasks as necessary to achieve the aim. In this way three basic types of analysis have to be emphasised. The first analysis is **validation**. This form of appraisal for example audits whether the workflow behaves as expected. This is often done by an interactive simulation, where fictitious cases are fed to the system. The second analysis is **verification**. Here e.g. the correctness of workflow is established. The **performance** analysis tries to find bottlenecks and to bypass them. The last two analyses need advanced techniques to fulfil their requirements. Several of these have been implemented for Petri nets (chapter 3.1) [Aal98].

Due to the needed data for analysis, workflow events have to be stored. The prototypical system EvE implements event-driven workflow executions by using an event-history following concept. EvE logs several event attributes like the event's type, the workflow where the event occurs, the timestamp when the EvE-server detected the event etc. for post-mortem analysis [GeTo97].

#### 2.5. Workflow Recovery

Workflow *recovery* enfoldes at least two features. First, undo uncompleted tasks that failed or even completed tasks that need to be cancelled. Second, redo an undone workflow. If a process is cancelled before it is done, two reactions are possible: On the one hand the workflow can be completed and a separate workflow can cancel the process, on the other hand the workflow can abandon and executed tasks can be undone. The first option is inoperative, because of four reasons [GHS95]:

- A workflow that will be undone after its completion requires **additional computing time**.
- **Human resources** are dissipated if installations are demolished immediately after their deployment
- A **second workflow** for destructing the process of the first one lavishes resources
- **Resources of installed facilities** cannot be used in other workflows for the time being

Evidently, relating to these reasons the second option is the better choice. So workflow management systems have to implement a function to cancel workflows and tasks during their execution to save resources and to guarantee the workflow's consistence.

### 3. Representation Concepts

Workflows need to be modelled. This chapter presents different modelling techniques with their advantages and disadvantages. First the most popular technique, *Petri nets*, will be explained by presenting several kinds of Petri nets (*normal*, *reactive*, *priority*, *timed*, *coloured*, and *hierarchical*). Subsequent a general survey of *process modelling languages* (PML) and several *UML* techniques will be given. Concluding, the differences of these modelling types will be discussed.

#### 3.1. Petri Nets

As aforementioned, Petri nets are able to model states, events, conditions, synchronization, parallelism, choice, and iteration close to the real processes and independent of their complexity. In this way Petri nets are the basic technique for modelling workflows, on the

one hand because of their deep theoretical background, one the other hand because of the simplicity to visualise the workflow with Petri nets. Former rests on five advantages [Kno00][ EsWi03]:

- **Provability** of a Petri nets' 'correctness'
- Adequacy for representing **discrete dynamic models**
- Existence of a large number of **modelling and simulation tools**
- Implementation of a Petri net based Workflow is **convenient**
- **Standardisation** of workflow management systems with Petri nets

The basic idea of combining workflows and Petri nets is in associating tasks with Petri nets' transitions, so that an executing task is equivalent to a *firing transition* [Kno00]. The problem herein is that workflow engines are *reactive* systems that must fire, while the token-game semantic of Petri nets may [EsDe03]. Due to this fact normal Petri nets are not as beneficial as they appear to be. Hence the literature presents other kinds of Petri nets that more competent in this requirement. Some of the most important will be described now.

A **reactive Petri** net can simply be build by changing the may-fire rule into must-fire rule. In this way the token-game semantic becomes reactive. Due to the closeness of Petri nets this proceeding is not advisable, because the system's environment is also included. The problem arises to the fact that the environment is active and not reactive. So, for the environment the may-fire rule and for internal transitions the must-fire rule is used. The behaviour of a reactive Petri net is described as follows. A Petri net has two possible states: stable and instable, whereas a stable state is reached if no internal transitions are enabled. If one or more environment's transitions are fired, the net becomes instable. In this case the system must continue to fire enabled internal transitions as long as it does not reach a stable state. This causes the problem that if the net contains a loop a stable state will be never reached and the system diverges. Therefore a reactive system moves from one to another stable state if it is free of loops.

Assigned to workflows, there are at least three characteristic that prevent normal workflow nets to be adequate for the workflow engine [EsDe03]:

- Transitions fire **instantly**.
- Transitions in a workflow usually **model tasks** unlike the workflow engine which monitors tasks but does not execute them.
- Workflows based on Petri nets are based on **active semantic**, whereas a workflow engine has to be reactive

These can be changed by refining tasks, distinguishing between workflow engine and environment, and modifying the firing rule.

Similar to reactive nets the so called **priority nets** were developed. This kind of Petri nets defines a static priority for each transition which orders them. These priorities obtain the liveness which means the net will not deadlock or produce any threads/situations that can not run [EsDe03][Bau96].

Based on the groundwork several new problems like mapping time and data or large models with so called **high level Petri nets** were discussed. Three of these most popular enhancements are the extension with colour to model data, the extension with time, and finally the extension with hierarchy for mapping large models.

**Coloured Petri nets** differ from a normal net in the way that every token has a colour. This not inevitably means a colour in the common sense but rather refers to the possibility of storing values in tokens. These can be altered by transactions.

**Timed Petri nets** are used to simulate the temporal behaviour of a system. This can be accomplished by using token, places, and/or transitions.

**Petri nets with hierarchy** are created by the use of subnets. These consist of places, transitions and subsystems. In this way a hierarchical structure is created. The top-level describes a process in the simplest possible way and any deeper level specifies the upper.

[Aal98]

### 3.2. PML

The PML, short form of **process modelling language**, has the ability to describe processes. Unfortunately there is no standard for PML but rather numerous dialects under the same name. So *the* PML does not exist. Due to this fact, this chapter will give general survey of PMLs. First a PML-developer has to decide whether the language has to be formal, semi-formal or even informal. Second the decision about the language “distribution“ has to be taken:

- One small kernel PML and several sub-PMLs for each sub-domain of a process or one huge PML, with sub-PML as view.
- One PML for all meta-process phases or one PML for each individual meta-phase.

The decision for a core PML is supported by several reasons:

- Actively **reuse** technology, so that conceptual and technical work does not have to be done twice.
- **Standardisation** (reduces costs of adaption)
- **Interoperability** with other processes
- Easy **user-level evolution** of the process model (one language causes no further costs for learning if an other process needs to be described) [CoJa03]

### 3.3. UML

While many PMLs are based on Petri nets, this chapter presents a completely different approach using UML. This language offers several diagrams for different process views [Hru98]:

- **Static structure diagram** for mapping team structure ,
- **sequence diagram** for mapping the instance of the business process,
- **use case diagram** for mapping static relationships between business processes,
- **sequence diagrams** for mapping interactions between business processes and actors,
- **collaboration diagram** for mapping interactions and relationships between business processes and actors, and
- **activity diagram** for mapping allowable order of business processes.

The latter, which is a special form of UML state diagrams, is worth a closer look. The first and most important difference to normal Petri nets is that a semantic for activity diagrams is reactive. An UML activity diagram describes the behaviour of a workflow system. This system has two main characteristics. First it is obviously reactive and runs in parallel with its environment. It responds or reacts to input events. Second a system has a co-ordination functionality which indicates that it does not execute a task but coordinates the execution by the agent. [EsWi03][DuHo01]

### 3.4. Comparison

As mentioned in chapter 3.1 normal Petri nets are active not reactive, they can not store data or time and are not adequate for large processes. For these kinds of requirements several Petri net extensions are necessary. In contrast to this UML activity diagrams include all these features. By virtue of the fact that reactive nets can not contain data (therefore coloured nets are needed) conflicting decision transactions are activated and the net behaves non-deterministic, whereas activity diagrams are deterministic. [EsDe03] [EsWi03] Concluding both, UML and Petri nets are in the position to handle all kinds of process requirements. PML can do this too, but due to the non-standardisation of this language several communication problems in collaboration with other systems may occur. Ignoring this problem, none of the presented approaches is more appropriate than the other relating to the technical potentialities and the choice is left to the personal preferences of the developer.

## 4. Agent Framework

As illustrated in chapter 2, modern agile workflow systems require a lot of flexibility i.e. the capability to react to changes in the workflow during its execution. Conventional software reaches the end of its capabilities here, requiring mostly predetermined paths. While this problem could be solved by using an interpreter for processing the workflow definition, many business processes are distributed over various workstations, departments or even organizations. In most cases compatibility across these boundaries is not given, making it impossible for a single workflow system to fulfil its task. Several studies see the solution to this problem in the application of *agent technology*. Agent systems are by nature distributed, consist of several autonomous entities, each of them possibly developed independently, and still manage to communicate in order to achieve a common goal. The following chapters take a look at the basics of *agent systems*, alternative *society* models, *communication languages* as well as possible appliances besides a workflow system. Agent theory has been a major research issue in the past years bringing up several different approaches, definitions and standards. This paper attempts to present a review of the general architectural decisions to be made when designing a framework but abjures giving technological backgrounds which can be found in other more specific articles.

Before proceeding with societies there is need for a definition of a single agent. An *agent* is an autonomous software unit which acts independently on behalf of its principal. While the above definition explicitly states an agent to be software it is important to note that in certain environments (especially workflow systems) an agent can often refer to both software and human agents. For reasons of precision one might distinguish between software and human agents.

Depending on the use of the term, there are several attributes assigned to agents. A common classification is weak versus strong agents. A *weak agent* is characterized by:

- **autonomy**: independent acting, based on one's own reasoning
- **social ability**: the ability to interact with other agents
- **reactivity**: the ability to perceive one's environment and react accordingly
- **proactivity**: not only reaction to the environment but the attempt to actively influence it
- **goal-orientedness**: the agent works for a goal set by the principal (agent or user)

*Strong agents* have additional attributes:

- **mobility**: the ability to move around in a network (across workstations, networks etc.)

- **benevolence:** the assumption that agents do not have conflicting goals, allowing them to work without having to solve conflicts first
- **rationality:** agents attempt to find the best way to achieve their goals
- **adaptivity:** ability to adapt to the principal's preferences
- **preservation:** ability to preserve the internal state from deactivation to reactivation

The list of attributes may not be complete but it contains the most important aspects for this kind of classification.

A different approach is described by *BDI-agents* (Beliefs-Desires-Intentions) which attempt to further specify the social and rational components of an agent. An agent is equipped with a set of beliefs describing its understanding of his environment (possible states and developments). A subset of all possible states is regarded as desirable (desires). They may be reached either through certain actions taken by the agent or by other circumstances. Sometimes the term "goal" is used to describe those states that can be reached based on the agent's own efforts. The term intentions finally addresses a set of actions the agent intends to take in order to achieve one or more of his aims. A sequential list of actions is often referred to as a "plan".

Regarding the implementation of the above concepts there are several suggestions, e.g. a layer-architecture [KKPS00]. Events or messages are received by the topmost layer and passed down to the bottom until they reach a level appropriate for processing. The top layer is the sensory level which is used to perceive changes in the environment or receive messages from other agents. From there, messages are passed down via a beliefs layer to reasoning, where the agent makes decisions on how to react and which actions to take next. Further layers are action, collaboration, translation and finally mobility. A different approach is given with a four component model. The agent should consist of four main modules, a communication manager, which is responsible for agent-to-agent as well as agent-to-institution communication, a contract manager containing the negotiation module (the authors describe an agent based trade-system) and the situation assessment module which holds the decision component. A task manager is responsible for planning further action [Dig03]. It is obvious that this classification is rather open, the decision component could include beliefs, desires and intention but it could also consist only of a simple logic-component.

Most of the above models see the agent as a dependent software unit working for a principal ("assist an end user") [Bla01]. Yet some consider this to be a very limited view and prefer to see agents as complete individuals. As such they could adapt their principle's goals but at the same time could have goals of their own making them possible unreliable partners in a transaction [DMWD02]. This is not a problem in controlled environments where the internal structure of the agents is predefined/known to the society, but more so in open environments like the internet.

#### 4.1. Agent Societies

A single agent by itself is of little use to the principal. There is a need to negotiate contracts, retrieve information or contact other agents. A collection of agents interacting with each other for some purpose and/or sharing a location can be referred to as a society. In recent years there has been a rising awareness to the similarities between human and *agent societies*. [CDDi02] states that "agents (like humans) can function more effectively in groups that are characterized by cooperation and division of labour." The same rules that apply to human societies also apply to agent societies: There is a demand for rules, services

and coordination, strongly depending on the type of society. One may distinguish between three basic types of **agent architectures**:

- **Single-agent** architecture: This scenario consists of only one agent that serves a predefined purpose. Common applications are user interfaces, or personal information agents. We will not regard this type any further for the remainder of this article.
- **Homogeneous** society: This type of society consists of numerous independent agents (possibly developed by independent authors) which all conform to a predefined interface. The main advantage of this setting is that even though each agent may have individual goals which are of no use to others they all share the same capabilities. So in theory it would be possible for any of these agents to fill roles currently required by this society (see chapter 4.3 for more details). Most platforms make use of this kind of architecture by publishing an interface and allowing any agent conformant with this interface to join the society. The interface most likely includes event-handlers, predefined messages etc. to allow for communication, creation and removal of the agents. Because of the independent development of agents there is already a chance for misbehaviour even though the agent meets the external requirements.
- **Heterogeneous** society: This is most liberal form of society. Agents are developed without restrictions and may enter or leave the society arbitrarily. Like human societies the only common capability has to be some form of communication. There are several attempts to standardize agent communication, some of which are presented in chapter 4.2. Other than this the agents are free to work for their own goals, form teams to achieve mutual goals or adapt any role they consider themselves appropriate for. As with all liberal societies this scenario is extremely vulnerable to misuse of any kind. There is no control over the agent's goals, their social behaviour in the beginning there are no instances taking control over the members of this society.

Strictly speaking, the heterogeneous society is the only fair open architecture. In past years many developers have designed frameworks for agents, found workarounds for conceptual insufficiencies and by doing so endangered or even destroyed any chance for new agents to become a part of that society. In other words a society can only be open to new members if there is no implicit knowledge required for interaction [EIAb04].

Facing the problems of coordination there are several attempts to merge agent-theory with the latest research in organizational theory. Before discussing alternative structures one must become aware of the situation the designers of such a society are in. When an agent enters the society it must not be his aim to work together with the other agents but rather pursue his own goals. Some authors state that in such a society the agents feel like "happening to be together" and do not especially care about the other agents' interests. An agent may react critical when asked to perform a service, have no sense of trust and in the beginning not be aware of any authority to rely on [DMWD02]. This must be taken into consideration in any kind of society but even more so when designing ecommerce or trade systems. Any kind of transaction or payment in these environments must be reliable, secure and valid. Taking the agent's personality into account, depending on developer and principal, one might find diverse social behaviour.

[DDD03] describe four basic types of agents:

- **Selfish**: when put in a certain role the agent attempts to fulfil this role but in case of conflicts with his own goals he prioritizes those

- **Social:** when a conflict between his own and society's goals appears the agent primarily remains in his role
- **Maximally social:** the agent devotes completely to the societies goals leaving his own aside
- **Maximally selfish:** the agent devotes completely to his own goals leaving the society's aside

For any kind of organization there are two contrary approaches, *bottom-up* and *top-down*. When forming the society top-down the designer begins with the creation of institutions, departments, necessary roles (see chapter 4.3) etc. and hereby predefines the shape of the society. Once the design is completed the agents may enter the organization and move to the available positions. The bottom-up approach starts with an unstructured group of agents with more or less common goals. Making use of their shared language they negotiate a structure best suited for their current situation and thus form the structure of the society.

The following section takes a look at different **forms of organization/coordination** adopted from the organizational theory [DWX03][Dig03]. An *organization* is defined as a set of otherwise autonomous entities working jointly to achieve common goals, regulated by a set of social rules and norms.

The traditional form of coordination is a **hierarchy**. Agents are organized in hierarchical groups, every group has a leader who himself may again be part of a group. The leader is responsible for the services his group has to offer, while most models see the actual way of working inside the group as irrelevant to anyone else but the members of this group. As in modern companies, the leader of the group is responsible for the results presented to the superiors. These groups of agents are sometimes referred to as "agencies". Chapter 4.3 takes a closer look at this concept. An interesting concept associated with hierarchies is that of *frame-contracts*. All agents either offer or need certain services, look to buy/sell a product etc. In hierarchies these kinds of transactions are usually managed by frame-contracts which mean a long term contract wherein all terms are fixed. The contract is "signed" when the two parties first meet within this organization and remains valid until either party wish to leave. This concept holds a major advantage: both partners can rely on the terms of service and do not need to renegotiate every time there is demand for a transaction. According to the theory of transaction costs this increases the contracting costs while at the same time significantly reducing the production costs (just in time delivery etc. is only possible if the buyer can rely on specifications, delivery and quality).

The opposite structure is a **market**. Here the organization basically has no structure. Instead the agents come together on a virtual market to negotiate terms of service, prices etc. Once a transaction is completed the contract is fulfilled and the agents move on to seek new partners. While this allows great flexibility it also makes transactions unreliable (neither party can rely on achieving its aim of buying or selling at a specific point in time). Transaction-partners as well as products change frequently.

The final coordination-concept found in agent-theory is a **network- or team-based** society. It slightly differs from a market-based structure where the agents simply meet, exchange products or information and leave to search for other opportunities. A network is usually formed by a group of agents who discover mutual goals which they can not or very ineffectively achieve on their own. Therefore they come together and form a society in which every agent moves into a newly defined role. Transactions among the group-members mostly serve the group's benefit instead of the individual agents.

It can easily be noticed that the concepts presented here merely differ from the description found in organizational theory. There is consensus regarding the fact that cyber-societies can best be understood and developed if they are modelled according to their human counterparts [DiDi04].

Workflow systems are traditionally hierarchical systems (there are assignments and authorities to report to). With the development of weakly structured or ad-hoc workflows there is a tendency towards network based systems in such a way that the agents share the common goal to complete a workflow and organize/contribute accordingly [TSP04].

Before proceeding, one should compare different aspects of the three architectures in terms of agent behaviour, negotiations etc. [DiDi04][ DWX03].

	<b>Hierarchy</b>	<b>Market</b>	<b>Network/Team</b>
<b>Environment</b>	closed to agents, open to data (new agents can not simply enter)	open (agents are free to join/leave)	closed (agents must have permission to enter)
<b>Relationship</b>	authority/superiority	competition	trust, mutual interests
<b>Purpose</b>	production of goods, knowledge or services	exchange of knowledge, goods or services	collaboration for a common goal
<b>Interaction</b>	predefined	based on standards	Individually negotiated
<b>Communication</b>	predefined routines	trade negotiations	shape society and build relationships
<b>Coordination</b>	supervision	price-mechanism	collaboration
<b>Goals</b>	global goal for society	mostly individual	global or individual
<b>Climate</b>	bureaucratic	suspicion, rivalry	mutual benefits
<b>Conflict Resolution</b>	supervision	haggling (negotiate own rules)	reciprocity/compromise

**Figure 1: Comparison of different coordination forms**

There are two aspects of figure 1 that require further attention. The first is the difference in agent attitudes. In a hierarchic society the agents know they can trust the other group members while society's requirements are obvious. The same is true for the network-organization in a less strict form. The agents have a common goal and may assume that there is neither treason nor selfishness involved in other agent's actions. In a market-based society on the other hand the agents are suspicious and must not in any way rely on the other agents. This leads to a high amount of mistrust, which can only be overcome by attempting to bring a form of legislation to the society (see below). Should one decide to remove beliefs, attitudes etc. from the agent (hereby stripping the personality to leave behind an intelligent algorithm) there is no need for further consideration of these problems.

The other important aspect is communication. It becomes obvious that the communication required in a hierarchic society is rather easy to implement. There are predefined interfaces, messages and events to react to and the agents basically move along predefined paths.

However it has been pointed out before that a common interface presents an unacceptable precondition.

One may conclude that the choice, which form of coordination to favour, is left to the designer. It strongly depends on the requirements and field of action. In an ecommerce-environment it is extremely important to allow arbitrary agents to join, communicate, form contracts and exchange services while in e.g. a workflow systems the requirements are very specific and may have to be precisely defined in form of an interface. In such cases, why should anyone bother and implement complex negotiation modules when all requirements can be predefined?

It may be worth noticing that in a certain way the different types of organization are similar but viewed from a different angle. Consider a hierarchical organization with atomic tasks at the bottom level. After cutting out a number of processes involved in a business process and putting them together as a group/department, the result is a team organization. Applying this to the complete hierarchy we receive numerous such groups that form a network. In modern organizational theory these networks should use market-mechanisms to determine appropriate prices for inter-group services. Eventually nothing has changed, either people work in a department or in groups, nevertheless they work on the same process, only the organizational pattern is slightly different.

So far it has only been shown that traditional coordination concepts can be applied to agent societies but it has not yet been described how they deal with the problems of communication, how they manage admittance, force the compliance with all contracts and fill vacant roles. Depending on the form of coordination a number of **institutions** help to solve these problems. Their general aim is to [DiDi04]:

- specify the **coordination structure**
- describe **exchange mechanisms** (kinds of contracts, price mechanisms etc.)
- determine the **forms of interaction** and communication
- **convey aims and norms**
- enforce the agents to subscribe to the **society's goals**

It is widely accepted that institutions enhance the efficiency of transactions and help to build trust. Still they are of less importance in market based systems. They can provide a trade-framework (standardized contracts, payment facilities etc.) or a common *ontology* to allow for better communication. Unfortunately at the same time they significantly reduce flexibility. There are predefined procedures that have to be obeyed. The market mechanism can not fully unfold [Dig03].

Contracts are an important means of regulation. On one hand individual contracts (agent to agent) contain the terms of transactions on the other hand there are social contracts (agent to society) which contain obligations and benefits of the agent joining the society. As with all contracts this type also needs some kind of monitoring and a legal authority responsible for penalizing agents in violation of these contracts (e.g. by expulsion or prohibition of further contracts) [DMWD02].

Several authors have developed society models with different institutions, some of which will be introduced now. Perhaps the most basic of all services concerns finding agents or services within a network. Agent-theory describes two kinds of institutions dealing with

this, *yellow pages* and *white pages*. Yellow pages categorize services without referring to specific agents. When in need of a certain service, e.g. information from a database, the agent contacts the yellow pages and asks for database-retrieval-agents. The yellow page service maintains a list of agents who are registered and offer these services and returns this list to the originator. The white pages on the other hand maintain a list of all registered agents regardless of their services. An agent may be aware that another agent exists and is willing to offer a required service but may not be aware where this agent is currently situated (in a network). To find out he sends a query to the white page service and receives the address.

In order to find an agent in white or yellow pages it must register first. Registration is done in two steps: Step one allows the agent to enter the society, in step two it registers with white-/yellow-pages. Step one requires an additional institution which is aware of the society's admittance policies and all open positions.

Every architecture uses different titles for the underlying institutions. As a reference this paper briefly introduces the architecture proposed by the *FIPA* (Foundation for Intelligent Physical Agents). This is a collection of organizations with the goal of standardizing intelligent agents including their attributes, communication and organization. A central aspect of this standard is its openness to independently developed agents and the ability to interact based on a common language (see chapter 4.2 for further details). The FIPA does not explicitly favour a hierarchical or market-based society in its standard but instead attempts to create a framework suitable to fit any form of organization. As a consequence the institutions described there are rather abstract concepts [*FIPA00*]:

- **Directory Facilitator (DF)**: contains the yellow page service; It is interesting to note that the architecture explicitly includes the possibility to have more than one DF in one agent platform, possibly requiring communication between DFs in order to locate a certain service not registered with the DF initially queried by the agent.
- **Agent Management System (AMS)**: this is the general management instance, responsible for creation and destruction of agents inside the platform. As such it is also best suited to take care of white pages.
- **Message Transportation Service (MTS)**: responsible for message transport inside the platform (agent to agent) and across different platforms.

It is interesting to note that this part of the specification does not include any details on trust centres, contracts etc.

There are several agent platforms available or in development and it is often hard to understand the actual roles/institutions required for the agents to function and those simply necessary for the specific framework application. With the basic services described this paper will turn to agent communication before describing the roles the agents can slip into. The concept of roles is to a certain extent similar to that of institutions (see chapter 4.3).

#### 4.2. Agent-Communication

Communication is essential for any distributed software. While the developers can still rely on common capabilities in homogeneous environments, heterogeneous environments are completely unknown to the developer. This poses two main problems to anybody willing to develop an agent compatible with this architecture (assuming that message-transportation is available):

- Which messages do the other agents understand, what are their replies?

- If both agents know message ‘x’, who can ensure that they have the same meaning?

There are different approaches to these problems, from extremely restrictive to extremely liberal. In the most restrictive version the communication-standard is set independently and in no way associated with the agent-society. It consists of a defined list of terms and *messages* without any possibility of interpretation. Every possible message has a defined syntax and meaning. There is virtually no possibility to extend this communication with custom or new messages without losing compatibility. The most liberal version relies on practically no fixed messages. The agents send arbitrary messages built from a standardized grammar according to an ontology the opposing agent hopefully understands. This poses a great challenge to any developer as he can rely basically on nothing. The most effective solution seems to be a combination of both approaches. Using a standardized set of messages (registration, notification, query etc.) and an exchangeable ontology this solution can be easily adapted to new situations. The agent receives a message it understands and adopts the ontology listed in that message. By doing so it has “learned” the new language and can communicate with the agent without requiring additional programming or conformity to an interface provided by the society.

By enabling the agent to learn new ontologies and therefore making it open to any kind of change, adding beliefs or other character trades poses no great challenge once they can be described by ontologies [DMWD02].

A different problem can be seen in the following description. If there are  $n$  agents in a society, speaking  $n$  different languages but willing to communicate, there are two possibilities: Either every agent has to be equipped with a translator for the remaining  $n-1$  languages (and any other possible language) or the society must agree on a common language. In the latter case all agents, whichever language they may otherwise use, must only include one translator from its own language to the common language, thus making them significantly smaller and easier to develop, especially taking into account that in an open society there are numerous unknown languages and/or *dialects* of which the developer is unaware at design time. However the common language would have to be predefined in some way [BCCJPRSG97][CDDi02].

The problem of translation between different languages or ontologies can e.g. be handled using a *thesaurus*. A group of mediator agents (see chapters 4.3, 4.4) could be used for translation if the other agents do not include a translation module [BBGGV03].

Some see ontologies as a severe limitation to the efficiency of agent communication. It is without a doubt a tremendous effort to fully specify an ontology. However, in the course of developing knowledge management systems many companies have developed thesauri for their domain which can be put to use in the agent-systems. For ecommerce there are several common formats for e.g. the exchange of product-specifications [CDDi02].

The ontologies pose a basic problem to any information/knowledge system. Some authors propose a three level architecture for optimal representation [ABHKS98]:

- Level three (bottom) is the **information ontology**. It is independent from the actual data stored in the ontology and stores primitive terms (e.g. author, document), data structures, access methods etc. This level can be reused in other domains as it is only a dictionary for frequently used terms with common definitions.
- Level two is the **domain ontology** which contains the actual content

- The topmost level is the **enterprise ontology**. It provides a context for the information stored in the domain ontology (information about the information) to allow for easier or more precise processing.

Regarding agent-based workflow systems some authors propose a separate ontology for the terminology required by the agents managing the workflow [TSP04].

It is hard to determine whether there is a best practice for the representation of ontologies but presumably it is efficient to separate domain-specific and general information. This reduces the size of the ontology (making it easier to send from one agent/institution to another) and allows the common part to be reused. Furthermore it could be managed and extended in a central institution from where the agents regularly receive updates and hereby “learn” new terms automatically. So far, to the authors’ knowledge, there is no such standard/institution available.

However some authors believe that the agents should not actually switch from one ontology to another, as their internal representation might be closely related to the original ontology. In these cases translation mechanisms must ensure a mapping from the internal to the external ontology. Currently research is looking for methods to let agents find appropriate translations, a job done by humans up to now. An interesting approach to learning a new language is described as pointing games: Both agents attempt to find objects they know and exchange their descriptions. By repeating this they slowly learn which translations to apply to their own terms in order to make them understandable to the opposing agent [ObMa03].

Regarding the previously described common language there have been several approaches, two of which will be introduced here.

The term used for these standards is **ACL** (Agent Communication Languages). ACLs are high level languages specially designed for negotiation, contracting, collaboration and exchange of information, the basic requirements of agent societies. They exist in a logical layer on top of common transportation protocols like TCP/IP, HTTP or CORBA’s IIOP. The latter is quite often used for agent technology as it allows language-independent interaction of software components including services like mobility and persistence. Aside from specifying the set of messages ACLs must also provide conversational policies, i.e. when to send which message or how to react [CDDi02].

Some basic requirements for an ACL are [Han02]:

- **Syntax** and **semantics** must be **expressive** enough to allow abstract interaction and enable problem solving
- The ACL must be **verifiable** to ensure correct messages
- Agent communication should be **independent** from the current domain

In an attempt to map human language to agent communication many ACLs are based on speech act theory. However developers tend to holding on to traditional programming concepts. This results in the sender preparing the message as a whole (accepting waiting times) and then sending it as one part. Some authors note that this is a breach of human language models where parts of a message can be sent as soon as they are ready [FIWi99].

The first attempt to standardize agent communication is *KQML* (Knowledge Query and Manipulation Language). It consists of a set of general purpose messages and a set of system messages used e.g. to register with a society. The available messages can easily be extended by creating new messages (see [TYS01] for an example). These must not meet any syntactical criteria. However this liberty causes KQML to quickly develop many incompatible dialects with unknown or ambiguous messages [Dig04a]. While the

convenience of the predefined messages makes this language easy to implement, the compatibility issues are a major weakness.

The previously mentioned FIPA has proposed the second major standard, the so called *FIPA-ACL*. The major difference between KQML and FIPA-ACL is that the latter does not have any system messages. It starts with a minimal number of messages and a set of syntactical rules. Any system messages have to be send using a combination of these general purpose messages. Extensibility is achieved by allowing the combination of existing messages under certain syntactical rules. If all agents obey the rules new messages can be understood by others without problems as they are easy to decompose into the basic messages. However the language module of a FIPA-compliant agent is much more complex than that of a KQML-agent. The FIPA's agent-specification adds further complexity to the language. As described at the beginning of chapter 4, agents can be conceptualized to have emotions or character. The FIPA-ACL has therefore been equipped with multi-modality, i.e. a message can have different "tones" or intentions although the message may nearly be the same [Dig04a]. In theory it is possible for an agent to make ironic comments. Even though this resembles human language it is to a certain extend doubtful if this is blessing or curse for agent-theory. From the developer's point of view there is no possibility to ensure that the opposing agent is capable of perceiving and processing these tones [MVB04].

Comparing both ACLs the choice once again comes down to the architecture's specific requirements. An open architecture or architectures otherwise using the FIPA agent model should favour the FIPA standard. Closed environments may use the KQML as it is much easier to implement.

The original drafts of FIPA-ACL and KQML had to battle a lack of interest because the concepts appeared to be too far off the actual business processes. Both were based on LISP which is not widely accepted for modelling business environments. In response to this more recent drafts are based on the widely accepted XML technology, including BRML (Business Rule Markup Language), to form an XML-dialect named ACML (Agent Communication Markup Language) [Bla02].

A next step in the development of ACLs might arise with the proceeding development of the semantic web, enabling the agents to "understand" the environment they are in.

While the actual language-standards are rather short the *interaction*\* protocols are by far more complicated. They deal with the problems of how to react to certain types of messages. A common classification distinguishes between *synchronous* and *asynchronous* messages. The former groups typical ask-reply or notify-confirm messages and is more or less easy to handle (unless there is high parallelism which may make it difficult to match reply and response). The second group consists of messages that require no immediate reply (e.g. tell). The opposing agent will not answer the message so there is no way of telling whether it has been received or accepted. Communication policies attempt to specify pre- and post-conditions to determine the situations in which such a message may be send (e.g. the sender must have the information and be sure that the opposing agent does not) and what situation to expect once the message has been received [com3apl.pdf].

---

\* [Sin03b] defines communication as preserving autonomy while interaction expects a reply/impact; the authors of this paper see the difference in single messages vs. message-sequences including pre- and post-conditions.

Besides defining the communication between two agents interaction protocols may attempt to formalize conferences or any other form of communication involving more than two agents. This implies keeping track of multiple messages, specifying when agents are allowed to speak and dealing with possible parallelism [Han02].

Unfortunately many developers see the present standards as a severe restriction and hence refuse to design systems accordingly [Han02][Sin03a].

While most authors still debate whether formal semantics or flexibility of an ACL is more important, some authors attempt to take the discussion to a different level [Sin03a]. It is not a goal of modern agent-standards to limit an agent's capabilities. Hence there are no rules on whether the agent should include mental states, goals, beliefs etc. The new proposal is not to attempt to standardize agent-to-agent-communication but to monitor the agents on a public level: When an agent joins a society it takes a certain role (e.g. seller of goods). In doing so it expresses its intention to conform to the society's rules and expectancies. In the further course of negotiations it is not important whether or not the agent believes the messages it sends but rather if its behaviour matches the expected role. So the basic idea is that communication should not be seen from the point of the individual agent but from the use it has for the society. However it is doubtful whether this theory solves problems on the communication level: What good is measuring an agent's role conformity if misunderstandings prevent the agent from telling the society that he is not capable of taking this role?

[PeE103] point out an additional problem often not considered in the interaction protocols: The common message "not-understood" can occur for several reasons. One of them is a message actually not understood or refused by the receiver. A second could be content which the receiving agent does not agree with or refuses to accept. The former is rather easy to detect by matching the message to the syntax-definition or the agent's role-expectancies. Detecting invalid or unacceptable contents is by far more complicated. They may e.g. occur because the agent can not understand why his opponent would send such a message based on the facts he is aware of. It may require additional research to specify protocols and ontologies that allow agents to precisely express which part of the original message they have "not-understood". After all there is no sense in sending the same message again (it was not a transportation error). Instead the sender must process the information he received in return to his original message and adapt his actions accordingly.

Before beginning with the implementation of an agent the above problems should be carefully considered. They may have a fundamental impact on the design of the framework's communication-component. Possible approaches are event-based communication (each agent is equipped with one or more event-handlers) or mailboxes (where the agent looks for new messages).

### **4.3. Agent Organization**

In agent societies there are two central concepts which intend to ease modelling and coordination. As with most organizational topics (see 4.1) these are borrowed from human societies as well.

The first concept is that of **roles**. *Roles* may be defined as patterns of behaviour. In a social system they describe the expected and externally perceivable behaviour of one or more members [DiDi04].

Although agent theory focuses on software which works uniformly in general one should keep in mind that BDI or otherwise “emotional” agents may behave differently in one and the same role (for reasons of qualification, motivation etc.). In severe cases agents may even be in violation of their roles. The society (chapter 4.1) must provide mechanisms to handle this kind of behaviour [DMWD02].

An agent society offers numerous roles, mostly depending on the domain it is set up in. Several agent societies have been developed each using its own nomenclature. Some basic services have already been described in chapter 4.1 (yellow pages, white pages etc.). They can either be implemented as static program instances or using agents filling the role. The following list contains the most common roles in agent-based workflow or knowledge-management systems [Dig04b][BCCJPRSG97][FIWi99][FWM03][Wan00][BBGGV03][Dig03][DiDi04]:

- *Gatekeeper / Domain Manager (KAoS) / Area Coordinator*: Depending on the form of coordination or the desired openness, the society may require an agent or institution in charge of admittance. The Gatekeeper takes on this responsibility allowing agents to enter or leave and maintaining a list of agents currently in the system.
- *Matchmaker (KAoS) / Yellow Page Agents / Role Coordinator (WONDER)*: Agents in this role are responsible for maintaining or accessing lists of services offered by the agents in a society. They may be queried to locate a specific service and are able to name agents capable of providing it.
- *Proxy-Agent (KAoS) / Cooperation Domain Agent*: Agents responsible for intra-society communication (agent-to-agent). These may be needed when e.g. translation is required (chapter 4.2) or agents are distributed and cannot establish direct contact. The proxy-agents receive messages, possibly process and relay them to the original destination. Moreover there may be occasions at which an agent needs to address numerous other agents. Instead of duplicating the message itself this job can be left to a specialized instance
- *Mediation Agent (KAoS) / Interaction Agent (CAGIS)*: These agents are responsible for inter-society or inter-workspace (CAGIS) communication. The distinction between intra- and inter-workspace communications is hard to make. One might see inter-workspace communication to take place between otherwise independent societies (ontology-translation etc. required) while intra-workspace communication simply deals with distributed environments of agents sharing one big society.
- *Transport Agent (KAoS)*: Agents responsible for other agents’ mobility in distributed environments. This may include persistence-services, backup- and serialization mechanisms etc.
- *Owner or seeker of information/products*: These are more abstract roles. The agents adopt them when entering a society with the aim of either selling or buying.
- *Banking authority, Broker (Can also be seen as institutions), Notary*: Commercial environments may require an authority that is trusted by both buyers and sellers to faithfully complete transactions of goods or finances.

Besides these more common roles any architecture brings along special roles. [BBGGV03] describe the MOMIS architecture which focuses on the integration of heterogeneous information sources. As such the system requires mechanisms known from relational

database systems like e.g. schedulers. Moreover the queries need to be mapped into the underlying system's own query language. These jobs are filled with *mapper* and *planner agents*. The actual query is executed by *query agents*. Merging the results from various sources requires further *management agents*. However it is doubtful whether these are abstract roles that can be filled by any agent joining a society or require specific programming. It is most doubtful that an agent without special capabilities and knowledge of the underlying system can act as a wrapper to legacy systems.

Agent-based workflow systems (e.g. WONDER [FWM03] or CAGIS [Wan00]) make use of a special *task-agent* or *work agent* which is responsible for the completion of a certain task (either itself or by monitoring software or users. In WONDER the task-agents (in fact the complete workflow instance) are coordinated by *case coordinators* and *process coordinators*.

A second important means of structuring agent societies is that of **agencies**. [norman97designing] define an *agency* as a consisting of a single responsible agent, a, possible empty, set of tasks that the responsible agent can execute and a, possible empty, set of sub-agencies represented by a single responsible agent.

Agencies offer certain services (e.g. banking). A *service* can be characterized as a collection of tasks with a predefined in- and output. To the outside it is only relevant that the output meets the defined criteria (e.g. payment is safe) but not how the tasks are processed internally or which sub-agencies and agents are used. From this point of view the agencies are autonomous units. Like a component in object-oriented programming these agencies can not simply be seen as subroutines. The actual agency can be exchanged like a component in a software system. Moreover just like components the agencies allow for easier modelling of the agent societies: Instead of having to define every single role the designers place an agency which can be independently designed and replaced at a later time. However unlike components which form the smallest unit of distributable software there are no such concepts for agencies. The agency can wrap around multiple sub-agencies in distributed environments. The same advantages that apply to components also apply to agencies: They may be reused, replaced or transferred as a whole, the outside specifications remain fixed. An agency can even offer services to other societies, e.g. an agency specialized in searching the World Wide Web may sell information to agents in a workflow system.

Finally one should note that the number of agencies and roles depends not only on the domain but also on the form of coordination. A market-based structure for example requires no gatekeeper or other admission-services. Since the agents look for independence they do not rely on the society to provide services but bring most them along by themselves. Still, the agents wish to avoid the danger of failed transactions and may therefore look for e.g. a banking authority or a notary to seal the contracts. Hierarchies or network-based societies on the other hand are often closed to outside agents and hence require gatekeepers, monitors etc. Since hierarchies are controlled, supervised environments with predefined contracts and long-term relationships there is less suspicion among the agents reducing the number of authorities required to provide trust.

#### 4.4. Where to use Agents

In modern software systems there are many fields of application for agents. It should be noted that the term agent does not always refer to a software-agent. Many systems address any kind of actor as an agent (agent vs. principal, user-agent etc.).

The best-known form of software agents are *Resource-Agents*. They are used whenever access to an external resource is needed, i.e. a hard-drive, data, the internet or a user. It should be noted, that not any user-interface is an agent, but rather agents can be user interfaces. To the agent “being” a user-interface is like accessing any other resource – it sends some information (visualization) and receives input (e.g. a mouse-click). This form of resource-agents is actually rather unimportant for agent-theory. A more fundamental appliance is the access to legacy systems. Specially designed agents have interfaces for existing software and the capability of translating queries and results to agents’ common language and ontology. These kinds of agents are often referred to as wrappers [JNFO00]. The concept sounds very helpful at first, especially in the context of workflow systems as described here that require access to various other programs. Unfortunately the agent-framework may not be compatible with the “old” technique. Agents are autonomous and hence in no way synchronized. Multiple agents’ need to access the same resource may arise at the same time. The external software may not be able to handle this load or not be ready for multi-user access and multi-threading (e.g. many word processors which do not support working on the same document at the same time) [JNFO00]. In these cases the resource-agent must also include a scheduling component, session mechanisms to simulate multi-threading and ensure that he is the sole resource-agent for this software.

Other widespread applications include [LHKR98][EIAb04]:

- *(Personal) Information Agent*: Agents that have access to a variety of information sources and are aware of their principal’s intentions and preferences. Their main goal is to access their sources and proactively provide information that is relevant to the user.
- *Domain Ontology Agent*: Responsible for managing ontologies, translating, storing and structuring organizational memory (knowledge)
- *Analysis Agents*: Receive information/documents for processing, categorizing etc.
- *Context-Provider Agents*: Make use of external sensors, domain knowledge etc. to provide e.g. information agents with information on their field of work. Using context-information can help agents to make better judgements on the information they find or the jobs the take.
- Workflow Management systems offer several opportunities to use agent technology (task execution, workflow management, stability etc.). Chapter 5 describes different architectures including agent-based techniques.

With all the possible applications in mind, one should not forget that there are some serious problems involved. Chapters 4.1 and 4.2 have already hinted at different attitudes, hidden intentions or the incompatibility of languages.

Another aspect concerns the problems of agent-only systems, especially in the context of business systems [YMS01][KRS00]:

- **Lack of coordination**: Agent-systems are distributed environments and offer virtually no control over the single agents unless some kind of hierarchy is established or there is a similar form of authority to which the agents are liable.
- **No Optimization**: The agent-systems are highly flexible, agent-based workflow systems attempt to deal with weakly-structured workflows (see chapter 2.2). The characteristics

- prevent all users involved from keeping track of the actual processes. Hence there is a good chance that the potential/necessity for optimization will not be used
- **Hard to Monitor:** The agents are autonomous and spread over various locations. There is nearly no way to keep track of the current state of work unless the agents at least report to some monitoring instance (see chapter 5 for details).
  - **Security aspects:** Any distributed environment needs communication. However in closed environments the protocols can be encrypted and remain unknown to a possible intruder. In open agent-societies one must be careful not to override security with compatibility. Modern security protocols e.g. Diffie-Hellman key exchange can help to take on these problems and find ways to encrypt communication between otherwise unknown agents.
  - **Legal aspects:** Agents are autonomous and do not undergo constant monitoring. This brings up the question of who is to be held liable for the agent's actions. Can the principal be liable even though he is not aware of the agent's actions?

## 5. System-Architectures

In the past years the architecture of workflow systems has significantly changed. While development started with one centralized architecture, there are several agent-based approaches available today (see below).

Before discussing alternative concepts the advantages of applying agent technology to workflow systems should be pointed out [*JNFOO00*]:

- **Flexibility:** Actions/task-execution can be based on the agent's current situation i.e. local circumstances (e.g. available resources etc.) as opposed to predefining them at design time
- **Agility:** new services/tasks may be added to the workflow without affecting other parts (agents). There is no need to lock the workflow or suspend/cancel its execution. The modifications only have local effects.
- **Adaptability:** As opposed to traditional concepts the agents may move along weakly structured-workflows (see chapter 2.2). This concept explicitly allows the workflow to be changed during or after execution e.g. as part of a learning process. In previous workflow systems these modifications required to change the actual workflow definitions. However in agent-based systems the workflow is dynamically read and transferred to the agents to allow easy modifications while at the same time the agents are enabled to modify the workflow based on feed-back (frequent branches, unreachable states etc.).

From a more technical point of view the advantages are [*LoZa01*]:

- **Component based** extensibility (depending on the architecture): Agents may be placed in different locations, combined, removed or exchanged at will. They do not require continuous access to a central workflow engine. Communication allows multiple agents to be combined in executing a single workflow instance.
- **Event/Exception-Handling:** Many mobile agents are written in JAVA, a programming language which is explicitly built on event- and exception handlers. Whenever an exception occurs during workflow execution or an agent needs to be notified these handlers can be used. To address the deficits in coordination the agent-system may use monitoring-agents with standardized event handlers to call and update the workflow status.
- **Remote Installation:** There is no need to install the workflow system, a present java-agent-framework can be deployed throughout the organization and used by the workflow system as well as other agent-based software. The agents can be sent to this remote location and commit work there.

- **Support for Mobile Devices:** Mobile devices are characterized by small memory and little cpu-power. Agents generally have a small code-base and carry only small payloads (the workflow) at the same time not requiring complex environments.

Traditional workflow systems rely on a central *workflow engine*. It is responsible for the execution of the single instance or multiple instances of the workflow. During the execution it locates the software required for the next task, sends the required data and receives the result after processing. This kind of architecture does not include agents at all. A first enhancement was to place software-agents at the end of each task, i.e. the software agent receives a load of work, interacts with the user or other software and reports the results back to the centralized workflow engine. This architecture is often referred to as *RPC-architecture*, originating in the RPC (remote procedure call) technology used for communication with the workflow engine. One can easily notice that the agents can not play out their full flexibility here. Their only advantage is the asynchronous work that does not have to be monitored by the workflow engine which is automatically notified by the agent on completion. However the workflow engine has proven to be a bottleneck which can not handle all the notifications, scheduling and task-assignments without a significant loss of performance as soon as the workflows become parallel or otherwise more complex. The next technological step is to relieve the central workflow engine by giving its job to different agents. The result is the *DartFlow Model*. When a new instance of a workflow is needed the workflow engine creates a new agent and equips it with all necessary data as well as the complete workflow. The individual agent looks at the next step in the workflow and makes a decision on where to execute it. Using its mobility-component it migrates to an appropriate system and executes the task (which can take a long time depending on the task, e.g. user interaction etc.). On completion it receives the result and stores it in its internal memory. Afterwards it moves to the next step, again decides where to execute it and migrates. Should the workflow fork the agent duplicates/clones itself and migrates to different locations. The workflow engine is reduced to creating the agents and monitoring the execution status (using event handlers invoked by the agents).

The third architecture is the so called *Maximal Sequence Model*. A sequence path is a collection of tasks that can be executed sequentially without splits or joins. If the sequence path cannot include another task to be executed sequentially, the path is called a maximal sequence path. Based on this definition the workflow is algorithmically analysed and split up into such maximal sequences. Every single sequence is assigned to an independent agent who can immediately begin work on the sequence (unless it must wait for certain input). The workflow engine's job is basically the same as in the DartFlow Model. In addition the engine is responsible for splitting the workflow.

Apparently the three architecture's performances must differ. While the RPC-architecture has to battle the bottleneck-problem, the DartFlow Model has to cope with bloated agents. They carry around the complete workflow as well as all the required data, making the required migration a significant problem. The Maximal Sequence Model on the other hand splits one workflow into many parts that may or may not be independent. To check on this and ensure correctness there is a tremendous amount of communication between all the agents.

Evidently this is just theory, but there have been some test-implementations. The initial test-setup consists of a workflow with 100 tasks. In this constellation the costs of migration and communication by far outreach the problems of the workflow engine in dealing with all the management. Therefore RPC-model is the quickest. However when increasing the

number of tasks (1000) as well as length and number of parallel branches the picture changes. The workflow engine in the RPC-model is swamped with a lot of management tasks and the response-times are significantly higher than those of the DartFlow or Maximal Sequence architectures. Surprisingly in this setup the DartFlow model still slightly beats the Maximal Sequence Model even though the agents carry their payloads. A second set of tests focuses on this aspect. The context-size (information required to execute the tasks) is increased and finally the Maximal Sequence Model shows its advantages. The communication effort is higher than that of the Dart Flow Model but still more efficient than constantly migrating large agents. The agents themselves are more efficient as well since the assigned part of the workflow requires less context-information and hence allows quicker access. [ABE03][YLSL03]

The *WONDER* workflow architecture is also based on agents and as such has to deal with the common problems of coordination and security (see chapter 4.4). One of the problems addressed explicitly is that of the loss of agents. Backing up a single workflow system is merely a matter of CPU-power or available memory. However the agent-systems are distributed, bringing up new problems e.g. what to do when the connection is lost or the agent disappears. In *WONDER* every agent clones itself on completion of a task in the workflow immediately before moving to a new location. It notifies a task coordinator (which keeps track of the current workflow status) of its current state. Should the agent encounter problems the coordinator can look up the last location and simply restart the workflow at the point it failed but preserving all work done so far. The backups are sent to a central server whenever unused bandwidth is available relieving the workstations of the additional data.

The article also describes a prototypical implementation of *WONDER* which is based on JAVA and CORBA. Performance tests show that the JAVA reflection mechanisms are very time consuming while actual use of the agents, messaging etc. has no significant influence on the system's performance [FWM03].

Some authors present a more integrative approach. Instead of (re-)developing agent based workflow systems they propose to add an *agent layer* to the existing systems. The agents are to form a safety net to handle failure conditions during workflow execution. In certain situations the normal workflow system might have to restart the complete process whereas agent technology could help in renegotiating and resuming the current instance. At the same time the agents could use a predefined interface to modify the fixed task-sequence adding or removing tasks as required. Finally agents could use their communicative abilities to translate data/procedure-calls from the workflow system's internal format to the format required by the external application or work as load balancers [OST99].

Further roles that could be taken by agents in a workflow system include [BIP99]:

- **Detour Monitoring:** A user of system may not be available. The agent should be aware of alternatives and could reroute the flow of work to the new location or even put aside the tasks, if possible.
- **Automated Work-Agents:** Agents that commit work not event- but time-based. Good examples are reminders or backup-jobs.
- **Advisors:** Agents that advise task-agents on how to do their job more efficiently. They process context-information (network bandwidth, processor load, user status etc.) and pass it on to the task-agent which can hereon base its decision where to proceed working.

There may be numerous other options depending on the specific architectures.

## 6. Collaboration and Proactivity

On several occasions the similarities between human and agent societies have been pointed out. As in human organizations **collaboration** is also a relevant in agent organization. There are jobs that can be done by one agent and jobs that are more efficiently done by more than one agent [GrKr98]. *Collaboration* can be defined as a joint intellectual or practical effort toward a goal [KBAV04].

Collaboration can be seen from different points of view. On one hand it allows entities to work more efficiently because they share resources and knowledge which would not be accessible otherwise. However there is often a negative touch to collaboration associating it to treason or working for the enemy. Even though for agent theory this perspective may seem somewhat extreme, it can not be denied that an agent collaborating may give more than it receives. Although a team effort should actually be judged by the results the team achieves there may be individual evaluations that do not fairly rate the agent's performance because it spent time assisting others hereby enhancing their evaluation [MACS02].

Collaboration can exist on several levels. The smallest is a team of agents working e.g. on completing a certain task in a workflow. Larger forms reach from agents managing a workflow up to agents that negotiate across organizational boundaries to conclude contracts.

Collaboration or any form of teamwork requires a common goal. Agent theory, especially BDI agents (see chapter 4), use the term plan to address a sequence of actions that eventually lead to a desirable state or achieve one of the agent's goals. The *SharedPlan*-concept extends this plan to a group of agents. The plan is shared by the team members which are all working parts of this plan to achieve the team's goal [GrKr98]. The authors note that plans are rarely complete but develop over time. In order to be precise the incomplete plans should be referred to as partial plans, making the team plan a partial SharedPlan.

An interesting concept for recurring collaboration is the so called *thinkLet*. A thinkLet is a named, package thinking activity that creates a predictable, repeatable pattern of collaboration among people working toward a goal. Teamwork requires coordination which should be as standardized as possible. Negotiating takes up a significant amount of time hereby reducing the efficiency of division of labour. ThinkLets present an interesting concept of standardizing repeating actions of collaboration [KBAV04].

When modelling agents capable of teamwork it is important to note that they require an additional component for conflict detection. The agents need to be aware of the other group members' work in order not to use resources required otherwise. The detection may include individual actions, i.e. an agent handicapping another, or group actions, i.e. the agents work has a negative influence on the teams overall goal [GrKr96].

This part of the paper describes possible collaboration patterns/dependencies. They reflect concepts known from most programming languages. Consider three agents A, B and C involved in some kind of collaborative process. [IOTA03]

- **Sequence:** Agent A completes a unit of work. Agent B waits for the results before beginning to work on its own tasks.
- **Function Definition** (no form of collaboration by itself, see below): An agent receives an input pattern, processes it in a task and produces a predefined output pattern.
- **Function Invocation:** Agent A places data in a function's input pattern. Agent B is responsible for this function, processes the data and stores the results in the output pattern. Agent A was suspended during the function's runtime and now receives the result to continue working.
- **If-Then-Else:** Agent A completes a unit of work and stores the results in an input pattern. Two agents B and C receive it and match it to an expected input pattern. Based on the results either of the two starts working while the other continues to wait for a new input.
- **Case:** Similar to If-Then-Else, more agents involved.
- **Loop:** An agent continuously works with itself. The results of one cycle form the input of a following cycle.
- **Parallel Fork:** Two agents start parallel work based on the same input pattern or an event
- **Synchronous:** An agent C merges the parallel forks of agents A and B as soon as both agents have delivered data required by the combined input pattern.

A second important issue in modern information/knowledge systems is **proactivity**. *Proactivity* as opposed to reactivity describes system behaviour where the owner of an information does not wait to be queried but actively notifies potentially interested other agents. The traditional reactive behaviour is sometimes referred to as master-slave paradigm [FaYe04]. Hereby the burden of determining whether or not information is relevant shifts from the one in possible demand to the one holding the information. This seems beneficial as the one holding the information is the only one who can find out that it is obsolete. With all the information available how can one keep track of its validity? It is rather the job of the source to inform other users of such information of its revision [YFV04].

Good proactive systems not only anticipate other's potential information need but actively attempt to expand the area of knowledge to obtain even more relevant data [Hol02][YFV02]. This may e.g. be achieved by actively querying other potential sources of information. The advantage in this kind of proceeding is that an agent A may obtain information from an agent C which it does not know, only because a queried agent B passed on the query.

However in order to be proactive an agent needs to be aware of the other agents as well as their possible information needs. Context-information plays an important role here. Relevant context information includes location, time, the agent's intentions, other activities, resources and data collected from physical sensors (temperature etc.) [SLS04].

The problem remains in how to determine possible interested agents. One can not keep track of every single other agent including context and possible information need. An alternative to knowing the other agents called subscription. The agents interested in information on a specific topic register with the information agents enabling them to maintain a small list of agents to track and notify. However this implies that agent A knows

that agent B will potentially discover relevant information and register. This may work if agents publicly accept the role of e.g. information agents (see chapter 4) but not if they have the information stored without explicitly offering it [YFV04]

## 7. Conclusion

This paper has presented the current state of the art in workflow modelling, agent societies and agent-based workflow systems. However, as this is a field of active research it is virtually impossible to present every detail. The authors have attempted to focus on the basic technologies and the most important architectural decisions as well as some common problems. When planning the implementation of software using related techniques it is extremely advisable to do further research regarding the specific aspects, as this paper is mainly focused on comparing architectures, not analyzing them in depth.

From the authors' point of view special research is necessary on the following aspects:

- **Proactivity:** Several authors mention proactivity as a basic requirement to information or knowledge-management systems. Chapter 6 has pointed out the advantages. However there is little information on models of proactivity. Recent papers attempt to formalize terms like information need or SharedPlans, still this only helps in representing possible interested parties. The actual notification and management concepts are missing. Chapter 6 has presented e.g. the subscription-concept but at the same time pointed out that either way all agents with potentially relevant information have to be known.
- **Context information:** Context information is essential to any proactive environment. At the same time it enhances the single agent's quality of work enabling it to decide on potentially attractive locations, relevant information or any other assistance it could require. Storing and retrieving context-information is similar to storing and retrieving knowledge in companies, similar mechanisms may be applied. However research has been unable to formalize human search strategies so far resulting in the inability to transfer the human concepts to agent-theory. Without doing so the agents remain inefficient in accessing context-information possibly missing relevant information just because they were not aware of other ways to access the information source. Besides storing the information the other interesting aspect is which information to store. Currently it is left to the designers which information the agents collect. In the long term the semantic web promises to change this. Software agents may finally be able to actively perceive and understand their environment allowing them to rationally decide which information to store and which to leave behind.
- **Agent Communication:** Different aspects of agent communication and organization have been discussed throughout this paper. However several steps remain before agent communication fully resembles human communication. The attempts to transfer speech-act theory to agent-communication are promising. However without further standardization the chances for fully compatible agents are small. Modern languages are based on XML, a technology which is widespread in businesses and software development and flexible enough to meet future requirements. Hopefully communication based on XML proves suitable for most developers to adopt otherwise the agent platforms remain incompatible. As globalization and the combination of different IT-systems proceed there is no room for proprietary concepts.

Regarding workflow modelling it is worth noting that both concepts (UML and Petri nets) are suitable. For PML there is currently no standard available. Some concepts use Petri nets, others focus on XML. In terms of agent communication and all efforts for standardization an XML standard would be desirable. The decision between UML and Petri nets is a matter of personal preference. Petri nets are a very traditional concept, numerous tools are available, and simulation algorithms can easily be found. Compared to UML the Petri nets can not be visualized quite as well as UML-diagrams. Regarding the usability for modern workflow systems both do not differ by much. However it should be mentioned, that modern business processes are highly dynamic and hard to specify. Unfortunately UML and Petri nets both require fully specified models (of the environment). This results in possible unspecified behaviour should any previously unspecified condition occur.

**Acknowledgement.** The authors acknowledge the European Commission for funding AMIRA under grant number FP6, project IST-2003-511740.

## References

[Aal98] vanderaalst98application.pdf

van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. In The Journal of Circuits, Systems and Computers, 8(1):21-66 (1998).

[Aal99] Nr18.pdf

Van der Aalst, W.M.P.: PROCESS-ORIENTED ARCHITECTURES FOR ELECTRONIC COMMERCE AND INTERORGANIZATIONAL WORKFLOW. Information Systems, Vol. 24, No. 8, pp. 639–671 (1999).

[Aal03] inheritance-of-business-processes.pdf

van der Aalst, W.M.P.: Inheritance of Business Processes: A Journey Visiting Four Notorious Problems. In: Ehrig, W. Reisig, G. Rozenberg, and H. Weber, editors, Petri Net Technology for Communication Based Systems, volume 2472 of Lecture Notes in Computer Science, pages 383-408. Springer-Verlag, Berlin/Heidelberg (2003).

[ABDEHH01] C46-IJCAI-DECOR.pdf

Abecker, A., Bernardi, A., Dioudis, S., van Elst, L., Herterich, R., Houy, C., Legal, M., Mentzas, G., Müller, S.: Workflow Embedded Organizational Memory Access: The DECOR Project. presented at JCAI'2001 Workshop on Knowledge Management and Organizational Memories, Seattle, Washington, USA (August 2001).

[ABE03] 003\_C2\_650\_abecker.pdf

Abecker, A., Bernardi, A., van Elst, L.: AGENT TECHNOLOGY FOR DISTRIBUTED ORGANIZATIONAL MEMORIES - The Frodo Project. In: ICEIS-2003 - 5th International Conference on Enterprise Information Systems, Angers, France. pages. 3-10. (April 2003)

**[ABHKS98]** OM-Technology.pdf

Abecker, A., Bernardi, A., Hinkelmann, K., Kühn, O., Sintek, M.: Toward a Technology for Organizational Memories. In IEEE Intelligent Systems, Vol. 13, Issue 3, pages 40-48 (May 1998)

**[AtHu00]** jcs00b.pdf

Atluri, V., Huang, W.-K.: A Petri Net Based Safety Analysis of Workflow Authorization Models. Journal of Computer Security, Volume 8, Issue 2/3 (2000)

**[Bau96]** prior\_nets.pdf

Bause, F.: On the analysis of Petri nets with static priorities. In Acta Informatica, Vol. 33, Nr. 7, pages 669 – 685, . Springer-Verlag, Heidelberg (October 1996).

**[BBGGV03]** Miks.pdf

Beneventano1, D., Bergamaschi1, S., Gelati1, G., Guerra1, F., Vincini, M.: MIKS: an Agent Framework supporting information access and integration. In Lecture Notes in Computer Science: Intelligent Information Agents: The AgentLink Perspective. Pages pages 22 – 49. Springer-Verlag Berlin/Heidelberg (July 2003)

**[BCCJPRSG97]** Agent Technology in Knowledge Management.pdf

Bradshaw, J.M., Carpenter, R., Crafnill, R., Jeffers, R., Poblete, L., Robinson, T., Sun, A., Gawdiak, Y., Bichindaritz, I., Sullivan, K.: Roles for Agent Technology in Knowledge Management: Examples from Applications in Aerospace and Medicine. In Proceedings of AAAI Spring Symposium on Artificial Intelligence in Knowledge Management pages 9ff, Menlo Park, California, USA (January 1997).

**[BIP99]** Workflow Management Systems Using Mobile Agents.pdf

Budimac, Z., Ivanović, M., Popović, A.: Workflow Management System Using Mobile Agents. In ADBIS'99, LNCS 1691, pages 168–178, Springer-Verlag, Berlin/Heidelberg (1999).

**[BKK04]** workflow oriented-system-for.pdf

Bassil, S., Keller, R., K., Kropf, P.: Workflow oriented System for the Management of Container Transportation: Challenges and Architecture. In Proceedings of the Second International Conference on Business Process Management (BPM'2004), Potsdam, Germany (June 2004). LNCS Springer-Verlag (June 2004).

**[Bla01]** Blake\_ICER4\_2001.pdf

Blake, M.B.: Agent-Based Workflow Configuration and Management of On-line Services. Proceedings of the International Conference on Electronic Commerce Research (ICECR-4), pages 567-588, Dallas, TX (November 2001).

**[Bla02]** Using-Agent-Control-and-Communication-in-a-Distributed-Workflow-Information-System.pdf

Blake, M.B.: Using Agent Control and Communication in a Distributed Workflow Information System. In R. CoopIS/DOA/ODBASE 2002, LNCS 2519, pages 163–178, Springer-Verlag Berlin/Heidelberg (2002).

**[CDDi02]** ciarticle1.pdf

Chaib-Draa, B., Dignum, F.: TRENDS IN AGENT COMMUNICATION LANGUAGE. In International Journal of Computational Intelligence , Vol. 18, Nr. 2 (2002).

**[CoJa03]** Process\_Modelling\_Languages.pdf

Conradi, R. Jaccheri, M. L.: Chapter 3, Process Modelling Languages. In Software Process: Principles, Methodology, and Technology, page 27ff, Springer-Verlag Berlin/Heidelberg (May 2003).

**[DiDi04]** Masta2001.pdf

Dignum, V., Dignum, F.: Modelling agent societies: co-ordination frameworks and institutions. In Lecture Notes in Computer Science Vol.: 2258 / 2001: Progress in Artificial Intelligence Knowledge Extraction, Multi-agent Systems, Logic Programming, and Constraint Solving : 10th Portuguese Conference on Artificial Intelligence, EPIA 2001, Porto, Portugal, December 17-20, 2001. Proceedings, pages 191ff. Springer-Verlag Berlin/Heidelberg (July 2003)

**[Dig03]** Amec2001.pdf

Dignum, F.: Agents, markets, institutions and protocols. In Lecture Notes in Computer Science, Volume 1991 / 2001, pages 98ff, Springer-Verlag, Heidelberg (June 2003)

**[Dig04a]**cia00.pdf

Dignum, F.: Agent Communication and Cooperative Information Agents. In Lecture Notes in Computer Science, Cooperative Information Agents IV - The Future of Information Agents in Cyberspace: 4th International Workshop, CIA 2000, Boston, MA, USA, July 7-9, 2000. Proceedings, pages 191 – 207. (February 2004)

**[Dig04b]** micai40123.pdf

Dignum, V.: Agent Support for Knowledge Sharing. Presented in MICAI'04, the Mexican International Conference in AI, Mexico City, (April 2004).

**[DDD03]** Aamas\_agent\_role\_7.pdf

Dastani, M., Dignum, V., Dignum, F.: Role-assignment in open agent societies. In International Conference on Autonomous Agents: Proceedings of the second international joint conference on Autonomous agents and multiagent systems, Melbourne, Australia, Session: Groups and organizations. pages: 489 – 496. ACM Press New York, NY, USA (2003).

**[DMWD02]** dignum\_final20829.pdf

Dignum, V., Meyer J.-J., Weigand, H., Dignum, F.: An Organization-oriented Model for Agent Societies. In Proceedings of International Workshop on Regulated Agent-Based Social Systems: Theories and Applications (RASTA'02), at AAMAS, Bologna, Italy (July 2002).

**[DuHo01]** dumas01uml.pdf

Dumas, M., ter Hofstede, A.H.M.: UML Activity Diagrams as a Workflow Specification Language. In Proc. of the 4th International Conference on the Unified Modeling Language (UML). Toronto, Canada (October 2001). Springer-Verlag, Berlin/Heidelberg, Germany (2001).

**[DWX03]** Agent-Societies – Towards-a-Framework-based-design.pdf

Dignum, V., Weigand, H., Xu, L.: Agent Societies: Towards Frameworks-Based Design. In Lecture Notes in Computer Science, Volume 2222 / 2002, page 33ff, Springer-Verlag, Heidelberg (August 2003).

**[ElAb04]** D00000694.pdf

van Elst, L., Abecker, A.: Agent-Mediated Knowledge Management. In International Symposium AMKM 2003, Stanford, CA, USA, March 24-26, Revised and Invited Papers. pages 1-30 (February 2004)

**[EsDe03]** eshuis03reactive.pdf

Eshuis, R. Dehnert, J.: Reactive Petri Nets for Workflow Modeling. In W.M.P. van der Aalst and E. Best, editors, Proc. 24th International Conference on the Applications and Theory of Petri Nets (ICATPN) (2003), volume 2679 of Lecture Notes in Computer Science, pages 296-315. Springer-Verlag, Berlin/Heidelberg, Germany (2003).

**[EsWi03]** comparing-petri-net-and.pdf

Eshuis, R., Wieringa, R.: Comparing Petri Net and Activity Diagram Variants for Workflow Modelling - A Quest for Reactive Petri Nets. In: Petri Net Technology for Communication-Based Systems, pages 321-351. Volume 2472 of Lecture Notes in Computer Science / Hartmut Ehrig, Wolfgang Reisig, Grzegorz Rozenberg and Herbert Weber (Eds.). Springer-Verlag (November 2003)

**[FaYe04]** teammodeling.pdf

Fan, X., Yen, J.: Modeling and Simulating Human Teamwork Behaviors Using Intelligent Agents. In Journal of Physics of Life Reviews (October 2004).

**[FIPA00]** SC0023K.pdf

FIPA TC B: FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS: FIPA Agent Management Specification. www.fipa.org (2000).

**[FIWi99]** KAW '99 Primitive Interaction Protocols for Agents in a Dynamic Environment.pdf

Flores, R. A., Wijngaards, N.: Primitive Interaction Protocols for Agents in a Dynamic Environment. In Twelfth Workshop on Knowledge Acquisition, Modeling and Management (October 1999).

**[FWM03]** ijcis.pdf

Filho, R. S. S., Wainer, J., Madeira, E. R. M.: A Fully Distributed Architecture for Large Scale Workflow Enactment. In International Journal of Cooperative Information Systems, Vol. 12, No. 4, pages 411-440 (2003).

**[GeTo97]** geppert97logging.pdf

Geppert, A., Tombros, D.: Logging and Post-Mortem Analysis of Workflow Executions based on Event History. Proc. 3rd Intl. Conf. on Rules in Database Systems (RIDS), Skövde, Sweden (June 1997), pages 67 - 82. Springer-Verlag, Berlin/Heidelberg, Germany (1997).

**[GHS95]** georgakopoulos95overview.pdf

Georgakopoulos, D., Hornick, M., Sheth, A.: An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. Distributed and Parallel Databases, An International Journal, 3, (1995).

**[GrKr96]** 20.pdf

Grosz, B. J., Kraus, S.: Collaborative Plans for Complex Group Actions. In Artificial Intelligence 86(2): 269-357 (1996)

**[GrKr98]** grosz98evolution.pdf

Grosz, B. J., Kraus, S.: The Evolution of SharedPlans. In Foundations and Theories of Rational. Springer-Verlag Berlin/Heidelberg (1998).

**[Han02]** Multi-agent System Infrastructure.pdf

Hannebauer, M.: Multi-agent System Infrastructure. In Autonomous Dynamic Reconfiguration, LNAI 2427, pages 95–113, Springer-Verlag (2002).

**[Hol02]** diss\_holz.pdf

Holz, H.: Process-Based Knowledge Management Support for Software Engineering (June 2002)

**[Hru98]** UMLWflow.pdf

Hruby, P.: Structuring Specification of Business Systems with UML (with an Emphasis on Workflow Management Systems). In OOPSLA 97-98 Published Proceedings p. 77ff, Springer-Verlag (1998).

**[IOTA03]** Flexible Multi-agent Collaboration Using Pattern Directed Message Collaboration of the Field Reactor Model

Iwao, T., Okada, M., Takada, Y., Amamiya, M.: Flexible Multi-agent Collaboration Using Pattern Directed Message Collaboration of Field Reactor Model. In Lecture Notes in Computer Science: Approaches to Intelligent Agents: Second Pacific Rim International Workshop on Multi-Agents, PRIMA'99, Kyoto, Japan, December 1999. Proceedings. Springer-Verlag Berlin/Heidelberg (July 2003).

**[JNFO00]** jennings00autonomous.pdf

Jennings, N. R., Norman, T. J., Faratin, P., O'Brien, P., Odger, B.: Autonomous Agents for Business Process Management. In International Journal of Applied Artificial Intelligence 14(2), pages 145-189 (2000).

**[KBAV04]** conceptualizationpaperDEF01.pdf

Kolfschoten, G.L., Briggs R.O., Appelman, J.H., de Vreede, G.-J.: ThinkLets as Building Blocks for Collaboration Processes: A Further Conceptualization. In Lecture notes in computer science, Springer-Verlag, Berlin/Heidelberg (2004)

**[KKPS00]** Kendall00java.pdf

Kendall, E. A., Krishna P.V.M., Pathak C.V., Suresh, C.B.: A Java Application Framework for Agent Based Systems. In ACM Computing Surveys Symposium on Application Frameworks. (2000)

**[Kno00]** P\_isd2000.pdf

Knorr, K.: WWW WORKFLOWS BASED ON PETRI NETS. In Proceedings of the 9th International Conference on Information Systems Development (ISD 2000), pp. 331-343, Kristiansand, Norway (August 2000).

**[KRS00]** Agents2000-knublauch-rose-sedelmayr.pdf

Knublauch, H., Rose, T., Sedlmayr, S.: Towards a Multi-Agents System for Pro-active Information Management in Anesthesia. In Proc. of the Agents-2000 Workshop on Autonomous Agents in Health Care, Barcelona, Spain (2000).

**[LaHa98]** labidi98collaborative.pdf

Labidi, S., Hammoudi, S.: Collaborative Workflow Management in WEICOT (1998).

**[LHKR98]** UM-CS-1998-052.pdf

Lesser, V., Horling, B., Klassner, F., Raja, A., Wagner, T., Zhang, S. XQ.: BIG: A Resource-Bounded Information Gathering and Decision Support Agent. In UMass Computer Science Technical Report (March 1998).

**[LoZa01]**

Towards\_distributed\_workflow\_enactment\_with\_itineraries\_and\_mobile\_agent\_management.pdf

Loke, S.W., Zaslavsky, A.: Towards Distributed Workflow Enactment with Itineraries and Mobile Agent Management. In E-Commerce Agents, LNAI 2033, pages. 283-294, Springer-Verlag Berlin/Heidelberg (2001).

**[MACS02]** amirati\_garg\_melian\_sevon.pdf

Melian, C., Ammirati, C.B., Garg, P.K., Sevón, G.: COLLABORATION AND OPENNESS IN LARGE CORPORATE SOFTWARE DEVELOPMENT. Presented at the European Academy of Management Conference, Stockholm, Sweden. With Catharina Melian, Cathy Ammirati, and Guje Sevón. (April 2002).

**[Mau01]** D00000547.pdf

Maus, H.: Workflow Context as a Means for Intelligent Information Support. In third International and Interdisciplinary Conference, CONTEXT (2001), Dundee, UK (2001). Springer LNAI 2116 (2001)

**[MeHe00]** adhocWF.pdf

Meng, J., Helal, S., Su, S.: An Ad-Hoc Workflow System Architecture Based on Mobile Agents and Rule-Based Processing. Las Vegas, Nevada, (June 2000)

**[MVB04]** Operational Semantics of a BDI Agent-Oriented Programming Language.pdf

Moreira, A.F., Vieira, R., Bordini, R.H.: Extending the Operational Semantics of a BDI Agent-Oriented Programming Language for Introducing Speech-Act Based Communication. In DALT 2003, LNAI 2990, pages 135–154, Springer-Verlag (2004).

[NJFM97] norman97designing.pdf

Norman, T. J. , Jennings, N.R., Faratin, P., Mamdani, E.H.: Designing and implementing a multi-agent architecture for business process management. In Intelligent Agents III: LNAI 1193, pages 261-275, Springer-Verlag, (1997).

[ObMa03] Mapping Between Ontologies.pdf

Obitko, M., Mařík, V.: Mapping between Ontologies in Agent Communication. In CEEMAS 2003, LNAI 2691, pages 191–203, Springer-Verlag, Berlin/Heidelberg (2003).

[OST99] ijcai\_99.pdf

Ogders, B.R., Shepherdson, J. W., Thompson, S.G.: Distributed Workflow Co-ordination by Proactive Software Agents. Proceedings of Intelligent Workflow and Process Management, IJCAI-99 Workshop (August 1999)

[PaMe03] knowledge-modelling-in-weakly.pdf

Papavassiliou, G., Mentzas, G.: Knowledge modelling in weakly-structured business processes. In Journal of Knowledge Management Vol. 7 Nr. 2. pages. 34 – 45. (2003)

[PeEl03] Ontology of Error Conditions.pdf

Petrinjak, A., Elio, R.: Understanding .Not-Understood.: Towards an Ontology of Error Conditions for Agent Communication. In AI 2003, LNAI 2671, pages. 383-399, Springer-Verlag, Berlin/Heidelberg (2003).

[Sin03a] Agent Communication Languages – Rethinking the Principles.pdf

Singh, M.P.: Agent Communication Languages: Rethinking the Principles. In Communications in Multiagent Systems. pages 37-50. Springer-Verlag, Heidelberg (September 2003).

[Sin03b] The Future of Agent Communication.pdf

Singh, M.P.The Future of Agent Communication. In Communications in Multiagent Systems, LNAI 2650, pages 318–322, Springer-Verlag, Berlin/Heidelberg (2003).

[SLS04] context\_aware\_ubiquitous\_services.pdf

Syukur, E., Loke, S.-W., Stanski, P.: A Policy Based Framework for Context Aware Ubiquitous Services. In EUC 2004, LNCS 3207, pages 346–355, Springer-Verlag Berlin/Heidelberg (2004).

[TaWa01] Agent-oriented\_Enterprise\_Modeling.pdf

Taveter, K., Wagner, G.: Agent-Oriented Enterprise Modeling Based on Business Rules. Proc. of 20th Int. Conf. on Conceptual Modeling (ER2001), Yokohama, Japan, (November 2001); LNCS 2224, Springer-Verlag, Berlin/Heidelberg, Germany (2001).

**[TSP04]** A Collaborative Multi-agent Based Workflow System.pdf

Tony, B., Savarimuthu, R. Purvis, M.: A Collaborative Multi-agent Based Workflow System. In Knowledge-Based Intelligent Information and Engineering Systems: 8th International Conference, KES 2004, Wellington, New Zealand, September 20-25, 2004, Proceedings, Part II, page 1187. Springer-Verlag, Heidelberg (October 2004)

**[TYS01]** Argument-Based Agent-System with KQML.pdf

Toda, Y., Yamashita, M., Sawamura, H.: An Argument-Based Agent System with KQML as an Agent Communication Language. In PRIMA 2001, LNAI 2132 , pages 48–62, Springer-Verlag Berlin/Heidelberg (2001).

**[Wan99]** wang99experience.pdf

Wang, A.I.: Experience paper: Using XML to implement a workflow tool. In Proceedings 3rd Annual IASTED International Conference Software Engineering and Applications (SEA'99), Scottsdale, Arizona, USA (October 1999).

**[Wan00]** icc200aiw.pdf

Wang, A. I.: Using software agents to support evolution of distributed workflow models. In International ICSC Congress on Intelligent Systems and Applications, Wollongong, Australia (December 2000)

**[WFMC96]** WFMC\_REF.pdf

Hollingsworth, H.: Workflow Management Coalition: The Workflow Reference Model. Future Strategies Inc., Lighthouse Point, FL, USA (June 1996).

**[WFMC99]** WfMC\_terminology\_glossary.pdf

WfMC: Workflow Management Coalition: Terminology & Glossary. Future Strategies Inc., Lighthouse Point, FL, USA (February 1999).

**[YFV02]** mcast-3.pdf

Yen, J., Fan, X., Volz, R. A.: On Proactive Delivery of Needed Information to Teammates. In Autonomous Agents and Multi-Agent Systems 2002 Workshop on Teamwork and Coalition Formation, Gologna, Italy (July 2002).

**[YFV04]** Proactive Communication in Agent Teamwork.pdf

Yen, J., Fan, X., Volz, R. A.: Proactive Communications in Agent Teamwork. In ACL 2003, LNAI 2922, pages 271–290, Springer-Verlag Berlin/Heidelberg (2004).

**[YLSL03]** Yoo01scalable.pdf

Yoo, J.-J., Lee, D., Suh, J.-H., Lee, D.-I.: Scalable Workflow System Model Based on Mobile Agents. In Lecture Notes in Computer Science: Intelligent Agents: Specification, Modeling, and Application : 4th Pacific Rim International Workshop on Multi-Agents, PRIMA 2001, Taipei, Taiwan, July 28-29, 2001. Proceedings, pages 222ff, Springer-Verlag Berlin/Heidelberg (June 2003).

**YMS01]** CSCWD01\_Maswfms.pdf

Yan, Y., Maamar, Z., Shen, W.: Integration of Workflow and Agent Technology for Business Process Management. In The Sixth International Conference on CSCW in Design. London, Ontario, Canada (2001).