Integrated Project
Priority 2.4.7
Semantic based knowledge systems

Information Society Technologies

The Social Semantic Desktop

NEPOMUK

Task Management Model

Deliverable D3.1

Version 1.0
29.01.2007
Dissemination level: PU

Nature	Report
Due date	31.12.2006
Lead contractor	SAP AG
Start date of project	01.01.2006
Duration	36 months

Authors

Olaf Grebner, SAP
Ernie Ong, SAP
Uwe Riss, SAP
Marko Brunzel, DFKI
Ansgar Bernardi, DFKI
Thomas Roth-Berghofer, DFKI

Mentors

Dimitris Apostolou, ICCS
Alexander Polonsky, COG

Project Co-ordinator

Dr. Ansgar Bernardi
German Research Center for Artificial Intelligence (DFKI) GmbH
Erwin-Schroedinger-Strasse (Building 57)
D 67663 Kaiserslautern
Germany
Email: bernardi@dfki.uni-kl.de, phone: +49 631 205 3582, fax: +49 631 205 4910

Partners

DEUTSCHES FORSCHUNGSZENTRUM FUER KUENSTLICHE INTELLIGENZ GMBH (DFKI)
IBM IRELAND PRODUCT DISTRIBUTION LIMITED (IBM)
SAP AG (SAP)
HEWLETT PACKARD GALWAY LTD (HPGL)
THALES S.A. (TRT)
PRC GROUP - THE MANAGEMENT HOUSE S.A. (PRC)
EDGE-IT S.A.R.L (EDG)
COGNIUM SYSTEMS S.A. (COG)
NATIONAL UNIVERSITY OF IRELAND, GALWAY (NTUA)
ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE (EPFL)
FORSCHUNGSZENTRUM INFORMATIK AN DER UNIVERSITAET KARLSRUHE (FZI)
GOTTFRIED WILHELM LEIBNIZ UNIVERSITAET HANNOVER (L3S)
INSTITUTE OF COMMUNICATION AND COMPUTER SYSTEMS (ICCS)
KUNGLIGA TEKNISKA HOEGSKOLAN (KTH)
UNIVERSITA DELLA SVIZZERA ITALIANA (USI)
IRION MANAGEMENT CONSULTING GMBH (IMC)

Versions

Version	Date	Reason
0.1	05.10.2006	First draft by Olaf Grebner
0.2	15.12.2006	Included several sections, e.g. State-of-the-art, Task operations, …
0.3	21.12.2006	Pre-Review-ready version for the mentors
0.4	08.01.2007	Integrated Nepomuk document template 1.4
0.9	16.01.2007	Review-ready version for the mentors
1.0	29.01.2007	Final Version

Explanations of abbreviations on front page

Nature
R: Report
P: Prototype
R/P: Report and Prototype
O: Other

Dissemination level
PU: Public
PP: Restricted to other FP6 participants
RE: Restricted to specified group
CO: Confidential, only for NEPOMUK partners

Executive Summary

The Social Semantic Desktop as developed in Nepomuk shall realize a comprehensive work environment for the knowledge worker.

This document describes the Task Management Model, which provides the fundamental representation for a structured modelling and handling of personal workload and individual activities. Motivated by our understanding of the semantic desktop as a predominantly *personal* tool, the task management model has to support a variety of usage scenarios, ranging from the personal ad-hoc annotation of an arbitrary information item as being somehow tied to an activity to-be-performed to the highly formalized representation of tasks as data elements within a distributed collaborative workflow application. This wide range of scenarios results in some guiding principles for the further development, which specify the expected benefits and the constraints to be considered, both from the individual and from the organisation's point of view.

Task modelling has been investigated for various purposes and from different viewpoints. Main approaches include the individual activity modelling, the coordination theory focussing on multiple actors, the unified activity management which generically represents human collaborative activities, and individual task and to-do list managers. A comprehensive review of the state of the art describes and discusses these approaches and coordinates with the requirements identified within the Nepomuk case studies.

Nepomuk's task management shall support a variety of operations on tasks. Personal aspects – annotating information as task-related, handling to-do lists, managing deadlines, structuring information resources etc. – are combined with and extended by organizational and collaboration aspects and evolved towards knowledge management aspects. The conceptual task management model describes the principle elements (concepts and relations) and functionalities considered by Nepomuk.

Finally, the Nepomuk task management model is defined as a set of concept definitions. Used as a data model (or domain ontology), this model offers the various data types and their properties which are needed to realize the intended functionalities. In accordance with the broad range of intended applications, most properties defined so far are considered optional, thus leaving room for very lightweight personal applications. On the other hand, further details and increased formality can always be realized by introducing new subclasses to the objects defined here.

In summary, the Nepomuk task management model offers the formal specification of the concepts and relations, which will allow to handle all data necessary to support task-related operations within the social semantic desktop, both in the personal as well as in interconnected and organizational settings.

Table of contents

1. Introduction

Nepomuk realizes the Social Semantic Desktop, which shall transform the computer into an effective tool for personal knowledge work. Central elements of personal knowledge work comprise information management, knowledge articulation, and sharing and exchange.

Besides these information-oriented aspects, the effective management of the personal workload is a fundamental challenge for any knowledge worker: Keeping an overview of the things to be done, structuring the daily activities as well as the long-term work packages of projects the worker is responsible for, observing deadlines, manage team cooperation, and report on timely achievements. These are some examples of the challenges any knowledge worker faces within the complex realm of multiple, parallel projects and activities which are typical for today's knowledge work. On the other hand, making a knowledge worker's tasks explicit provides a powerful tool for effective information structuring and handling.

Nepomuk takes care of this challenge by developing support for personal task management. Nepomuk Personal Task Management shall provide the necessary technical and methodological means to support the explicit definition, handling, and control of tasks within the personal knowledge work, both solitary or within team structures. The Nepomuk Personal Task Management Model, described in this document, shall facilitate the representation of all data necessary to represent knowledge worker's activities and to realize the support functionalities envisioned. This comprises

- the definition of a basic framework for task modelling, which allows for a sufficiently rich representation of tasks, taking into account the wide variety of possible interpretations; and
- the identification of task-related operations which are to be supported by the system

Building the representation formalisms of the Personal Task Management Model, and designing the intended support functionalities, takes into account a number of sources for requirements:

- Tools for task- or activity-management abound, ranging from 'to-do-list' –like tools on the personal computer up to organization-wide workflow management and process modelling systems
- A plethora of scientific literature has tackled important aspects of activity modelling and task management
- The case studies investigated within the Nepomuk project give insights into the domain-specific (and general) needs of knowledge workers in practice. WP10000 – Organizational Knowledge Management Case Study – in particular emphasizes the need for task management and work process support in a distributed, knowledge-intensive work setting within a large organization.

The common view on the various requirements and application scenarios which is taken by Nepomuk is the focus on the individual who manages the own tasks within the personal workspace. This personal task management is embedded in the conceptual and organizational work environment. The task-related operations need to cover the different scopes of collaboration and

sharing within this realm. Accordingly, the task representation formalism is tailored to accommodate these operations. The different scopes are

- the solitary work scenario: An individual knowledge worker follows some personal goal without any direct relation to the outside world. A typical task representation here is a simple to-do list (although all kinds of more structured representations and more detailed planning are possible). Operations comprise the initial naming of a task, its refinement even during the work, scheduling and reminding about deadlines, and the final note that some task has been completed. Extended functionalities may be oriented towards detailed log keeping, time recording, or other types of work documentation. Besides the usual advantages of calendar support, deadline reminder, personal time management, and work records, this scenario already offers the means to realize task-oriented and context-sensitive information management support.
- the team cooperation scenario: The individual is embedded into a team of collaborating persons. Within this team, tasks can be delegated or transferred, information is shared, and notes about the completion of a task are transmitted to other team members. With increasing size of teams and detailed definition of competencies and roles, this scenario expands to
- the organization scenario: Individuals are now embedded in well-established organizational structures with defined roles and formal responsibilities. Consequently, task-oriented operations take into account hierarchies of control, direction and delegation rights, and reporting obligations. Individual ad-hoc modelling of tasks is extended by formal pre-defined standard operating procedures, and roles are a new target of task transmission: Instead of directly addressing known individuals, a task might be given to an organizational entity or a role which then decides autonomously about the person who will work on the task.

Besides these varying degrees of responsibility, collaboration, control, and sharing, the modelling of tasks creates a data representation of work which can be transmitted over time and across individuals. From this viewpoint, personal task management unites future-oriented planning, present work execution, and the documentation of past activities. Furthermore, the task models can be stored, retrieved, and re-used, thus allowing for the transfer of experiences, the development of abstractions and generalizations, and ultimately contributing to the management of the know-how contained within the task descriptions.

Figure 1.1 shows an overview of the personal task management in these dimensions: In reality, work is planned (future), done (present) or a historical thing of the past. Personal task management supports these aspects by a variety of (future-oriented) planning and scheduling activities. Present work execution is supported by services for information handling, reminding, delegation and control. The preservation of task histories and work process trails results in case-specific and/or abstracted and generalized know-how which can be re-used at whim for new planning.

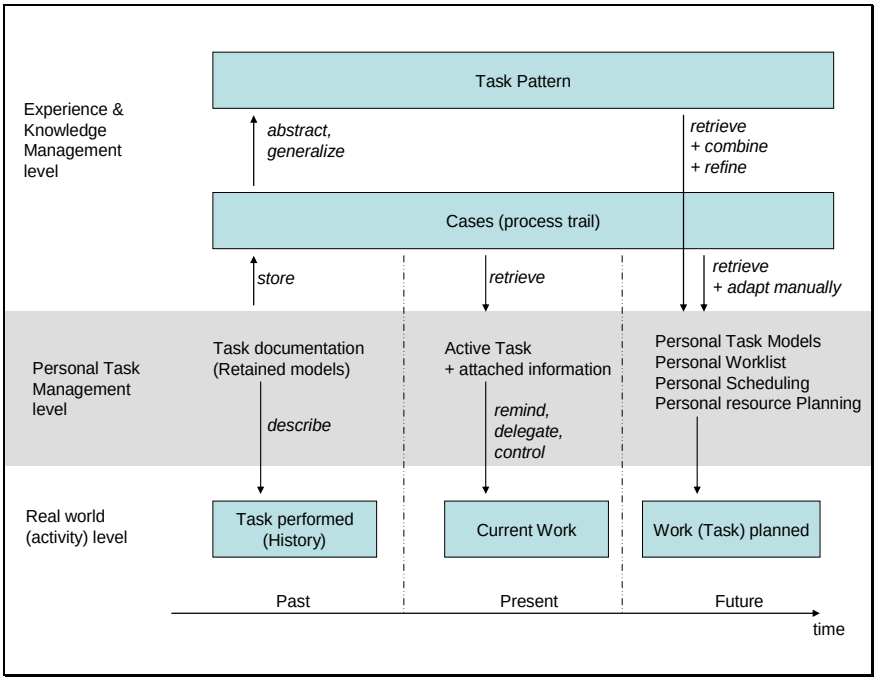

Figure 1.1: Dimensions of Personal Task Management

The representation formalisms and operations defined in this deliverable are oriented at these scenarios and allow to cover all possible instantiations.

2. Guiding Principles

Work package 3000 (WP3000) aims at realizing an ***integrated*** **task management support** for the individual knowledge worker in a networked environment. This means that it supports collaboration of the knowledge workers in the network with respect to related or dependent tasks. The leading principles of this task management can be described as follows:

- **Provide individual benefits by suitable services**: The Nepomuk task management provides more direct benefits to the users than it causes additional costs for them. Behind this requirement we find the insight that knowledge workers are rather delicate in the choice of tools that they use. Tools that do not clearly provide advantages will be rejected even if they might be preferable from an organisational point of view;
- **Provide social benefits by collaborative work**: The Nepomuk task management is to establish social benefits derived from individual experience based on task execution. Here we face the shared database problem: knowledge workers must provide information to others without a direct benefit for themselves. Therefore, to achieve this aim, the previous principle becomes especially important.
- **Provide organisational benefits using others' work experience**: This is related to the previous principle. However, whereas the social benefit aim at an indirect individual benefit, i.e., the individual user profits from the experience of others, organisational benefit might only indirectly lead to the knowledge workers benefit by making the organisation for which they work more efficient.
- **Ensure knowledge workers' autonomy**: The Nepomuk task management must respect the autonomy of knowledge workers. This means that the system must support knowledge workers in an unobtrusive way, providing guidance instead of prescription. Also here the question of motivating users to work with the task management comes to the fore. Moreover, the aspect of autonomy is closely related to the demand for flexibility of knowledge work processes;
- **Respect knowledge workers' privacy**: The Nepomuk task management must respect the privacy of knowledge workers by protected the individual work sphere. It must keep the balance between supporting the exchange of information between users and protection of those data that the users want to keep for themselves. Also this principle is to be seen as a necessary precondition to foster acceptance of the task management by the knowledge workers.

These principles are a precondition for the entire design of the task management. To realize the principle of individual benefit we realize an individual task management that is embedded in the user's personal desktop and has to tackle

- ad-hoc task planning and flexible changes (autonomy);
- collaborative work on task (social benefit);

- knowledge-intensive tasks with a huge amount of personal as well as group information objects (individual benefit), and
- integration into organizational processes (organisational benefit).

The functionality to be implemented concentrates on the structuring and management of personal work activities and collaborative aspects. On the one hand, the task management will be designed to handle the individual organisational needs of users, enabling a seamless transition between regularly used tools like email and notes and the personal tasks. On the other hand even personal task are related among each other resulting in a net of interrelated tasks. These nets must be organized and made transparent. Finally, the task management is integrated in the entire social semantic desktop and the architecture of the system will be adapted to this fact.

Moreover, the principle of autonomy suggests a task management system that is characterised by the sovereignty of knowledge workers over the execution of their tasks. Even if the a task is to be executed to fulfil the demands of another party, the task owners can freely decide in which way they want to execute their tasks. This requires the largest decoupling between tasks possible, reducing the mutual visibility between tasks and derived tasks to a degree that only allows for the natural interests of both parties.

On the other hand there are tasks that cannot be performed by a single person due to their complexity and requirement of various expertises. In this case the separation must be broken to enable a seamless cooperation of the included co-workers. Here the social benefit exceeds the desire for autonomy. This refers to the fact that collaborative team work is a common phenomenon today. In contrast to the individual case, such cooperation requires that information between the co-workers is as transparent as possible, i.e., that a common information space is established. Nevertheless, this shared information space is restricted to the well defined group of co-workers while other users are treated in a way that protects group privacy and autonomy. In this sense a task should be clearly related to the encapsulated activity of an individual task owner or a collaborating group.

3. State of the art analysis - task modelling

This section presents the state-of-the-art in task modelling. The goal is to provide a solid knowledge base for the modelling of tasks and task patterns as they presented in sections 5 and 6.

This state-of-the-art analysis includes a literature review of theories, on which a task management can be founded and which provide promising methodologies, as well as a review of relevant existing task models as these have been realized in different prototypes.

3.1. Task model methodologies

This section presents so-called task model methodologies. These methodologies describe the aspects to be considered in the development of the task model, i.e., it drives the modelling of tasks from a specific, strategic perspective. We refer to them as 'what-concepts' since they point at the problems and issues that task management has to tackle.

Regarding the modelling of tasks, there exist several methodologies, each having a predominant concept of abstraction, e.g. activity-centricity, user-centricity or process-centricity. All of these methodologies have a long history of successful applications and therefore provide the promising means for the development of a task management model.

Thus, we turn our attention to those aspects of these methodologies, which are relevant for personal task management. They help us understand the problems that stand behind the handling of tasks. First, Activity Theory (AT) focuses on the determination of activity between subject and object. Second, Coordination Theory (CT) highlights the management of dependencies between activities. Finally, Object-Oriented Activity Support presents an activity-centred perspective distinguishing activity types and instances.

3.1.1. Activity Theory

The principles of Activity Theory (AT) have been developed by Leontiev based on Vygotsky's cultural-historical psychology. Vygotsky's work in the 1920s aimed at understanding the interdependency of mind and society, resolving dichotomies such as those of mind and body, thought and action, or individual and society. His theory is centred on the idea that psychological processes are determined by the mediators which can be material or mental.

In particular, AT states that the relation between **subject** and **object** is determined by activities (Leontiev, 1978). The notion of activity is not restricted to human activities but can also be understood in a more general sense. It only assumes that the subject refers to an object in the world in a purposeful interaction. This interaction leads to a development of both, the subject as well as the object. While the object might be modified during the interaction leading to a state described as **outcome**, also the subject can undergo alterations, e.g., acquire certain capabilities, from the process. The asymmetry between subjects and objects results from the active role of the subject in the interaction caused by the subject's respective needs.

The interaction of subject and object is mostly not direct but mediated by mediating artefacts or **instruments**. These can encompass tools such as hammers as well as more symbolic tools such as pictures or language. Usually this relation is described in a triangular form of activity systems as depicted in Figure 3.1:

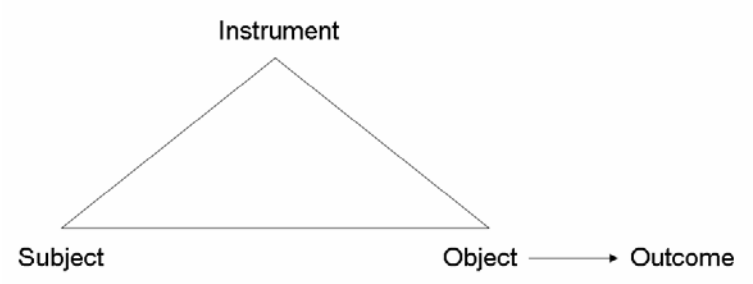

Figure 3.1: Engagement of subject toward objective mediated by instruments.

Regarding the instruments the distinction between physical and symbolic artefacts makes a decisive difference which becomes apparent in the phenomenon of **internalization**, e.g., frequently subjects stopped to use (external) symbolic artefacts due to the fact that they develop a specific routine in using them (Vygotsky, 1983). However, also the contrary process of **externalization** occurs, e.g., in cases of trouble shooting, and reflection. A central statement of AT is that activity always occurs in context. Consequently, if we examine human activity we have to include the motives, goals, and intentions of actors and which artefacts are involved in it.

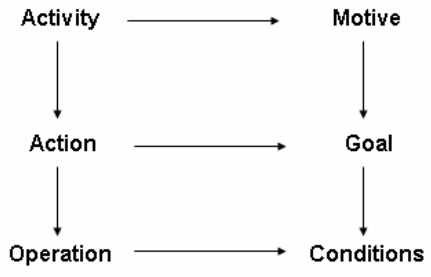

Figure 3.2: Hierarchical conception of activity.

An activity can be described as a three layered hierarchy as depicted in Figure 3.2 (Leontiev, 1978). In AT every interaction of a subject with the world can be described as an activity if it is motivated by a particular need of the subject. The motive stimulates the subject to perform the respective activity, even if often in an unconscious way. The motive can be represented by an object the subject strives for. However, there are cases in which the motive and the concrete object of an activity differ. In this case, we distinguish activity and action where the former is related to the motive while the latter is related to the object (Leontiev, 1981). For example, the goal of my action could be to write a book while my motive could be to acquire reputation by the publication of this book. Finally, actions can consist of operations. These are routine processes of which the goal might not be aware of anymore. In the sphere of knowledge, the distinction between actions and operation corresponds to the difference between explicit and implicit knowledge as described by Polanyi (1966).

Blackler (1995) states that knowledge is closely related to corresponding activity systems. He comprises the insights provided by AT regarding knowledge as follows:

1. Knowledge is **mediated**, e.g., by language, tools, symbols etc.
2. Knowledge is **situated**, i.e., depending on the context
3. Knowledge is **provisional**, i.e., it is constantly evolving in the context of its application
4. Knowledge is **pragmatic**, i.e., intentional and object-oriented

These are partially derived from respective categories for activity compiled by Engeström (1987) and thus reflect the close relation between knowledge and action.

The triangular description in Figure 3.2 does not yet include the dynamics that appear in a system of several actors. An adaptation of this kind has been provided by Engeström (1987) and is depicted in Figure 3.3.

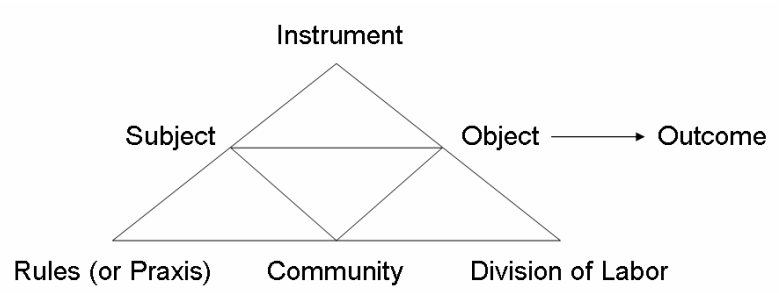

Figure 3.3: Structure of a human activity system.

This figure describes the collective activity of human actors. The **community** describes the group of actors the relationship of which is mediated by a common ground of meaning given by a common praxis (as formal rules or informal habits). The **rules** (or **praxis**) guide the collaboration of these actors. The relationship between the co-workers and the object of activity is mediated by the organization of work, described as **division of labour**. The co-workers find themselves in the community both as independent subjects and as object of interaction. These concepts have been mainly applied to identify requirements for system design that result from a work situation (Turner et al., 1999).

Referring to Engeström (1987) within this activity system four levels of contradictions can occur that accompany the change of activities. **Primary contradictions** are related to contradictions at singular node, e.g., the subject that might be driven by different motives. **Secondary contradictions** derive from the direct interaction of nodes, e.g., between subjects and tools which might not be appropriate to the subjects' goal. Contradictions between an activity and its more advanced form appear as **tertiary contradictions**. Finally, **quaternary contradictions** appear between different activities. The discovery of these contradictions can help to develop the activity system further, e.g., by introduction of improved instruments.

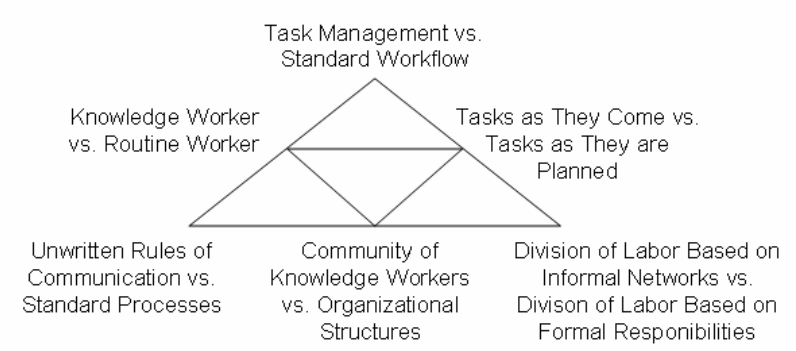

Figure 3.4: Contradictions in a network of human activity systems.

Figure 3.4 describes the primary contradictions for activities related to abstract knowledge work in analogy to the description given by Engeström (1987). These contradictions primarily result from the difference between informal processes and formal standards that are typical for organisations.

In the following, we want to investigate how AT can be used to derive requirements for a generic task management. To this end, we investigate the general role of the constituents in Figure 3.4 for the distribution of work. We refer to the investigation that has been proposed by Jonassen and Murphy (1998).

3.1.1.1 Clarification of the Purpose of the Activity System

We start with the division of labour since it concerns the central topic of the task management. Engeström (1997) pointed out that in contrast to the origins of collaborative work in families and tribes, in modern working life the relations between those who distribute the work and those who finally execute it can be rather broad. It is not even always necessary that they know each other. In this case, the object that is related to the division moves into the centre of interest. However, there are also cases where the community, e.g., the participant in a meeting work closely together so that their relationship plays a decisive role. In the former case, we often find a difference between motive and objective while in the latter case the participants are much more concerned with the success of the community so that motive and object move together to a large degree.

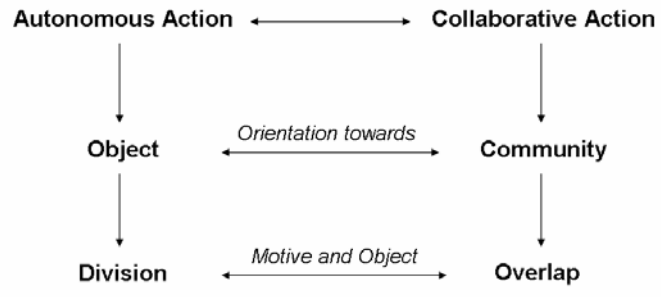

Figure 3.5: Autonomous vs. Collaborative Action.

Figure 3.5 shows the principle difference that appears between these two cases. Accordingly, the task management has to support both kinds of derived activities appropriately. While in the case of **Autonomous Action** the privacy of the actor must be preserved and the outcome

must be well specified, in the case of **Collaborative Action** the communication move into the centre of interest. We will consider this separation also in the following investigation.

Since we use the task concept in the sense of separable units of work, we will describe Autonomous Actions by different tasks while Collaborative Actions will be describes as one task to which several co-workers contribute. In the first case, we have to establish a formal communication channel, in the latter case we have to provide a collaborative working space.

3.1.1.2 Analysis of the Activity System

The analysis of the Activity system concerns the individual components as described in Figure 3.5 in order to get a clearer picture of their role. The distinction that we have made in the previous section will reappear also here in the descriptions of the different components.

3.1.1.2.1 Subjects

The subjects who work with the task management system are users of a system that work together with other users on tasks. Their motivations can be quite different and depends on the fact whether they belong to a organisation such as a company or to a loose network or users who share a common interest. However, this distinction does not concern the way how they organize their work. Whether actions are autonomous or collaborative mainly depends on the character of task and not on the attitude of the user among each other.

The outcome of an activity can be rather isolated, e.g., if a problem occurs and the user requires a solution as in the Mandriva case, or they can be rather structured as in an organisation where several people work on complex projects that require extensive coordination. The common footing of these activities is that all participants deliberately work on the tasks that they face to accomplish them successfully.

Users have different expertise and therefore concentrate on different aspects of a task or problem to be solved. The goal is to delegate some piece of work to that user who is expected to provide the best result. This might not also be the most experienced expert since it is often takes some time until they can work on a task so that a less experienced user who can answer earlier is often more appropriate.

3.1.1.2.2 Relevant Communities

The respective communities depend on the network in which the task management is used. Therefore, they can be rather different. However, there is always a certain willingness to support each other. The reason for this can be that the community is formed by a common idea as in the case of the Linux community or by a common corporate identity as in the SAP case. This supports the delegation of work, which is generally necessary, since individual users alone cannot execute complex tasks. In general, the accomplishment of a task in this network requires the support of several other users.

Depending on the particular community, the rules can be more formal, as in the organisational case, or more informal, as in the Linux community

case. However, since the focus of the task management is knowledge work the reach of formal structures is rather limited. This means that most communities are characterised by a joint understanding of the common goal.

Moreover, the situation is characterised by the fact that the willingness to cooperate with other users is noticed within the network and a crucial factor for mutual support. In the organisational case, the management can relate monetary benefits to the successful accomplishment of tasks. However, we must regard the role of social recognition as at least equally important (even in companies).

3.1.1.2.3 Objects

The outcome of an activity in the considered cases is generally an information object. Partially there are also services related to this information object, as for example in the case of a journey planning. There, the recipient does not only get the information about the booked flight, for example, but the entire flight service is related to this information.

Concerning the quality of the outcome there are several possibilities. On the one hand, we have tasks that are delegated. Here the delegating party expects a certain quality of the object that allows them to continue with their own activity. Event triggered task mainly arise from a state that has to be changed to another state, e.g., if a machines breaks and has to be repaired. The particular community generally knows and accepts these states. However, there are also conflicts if the respective expectations are not fulfilled.

The distinction between instrument and object is not always as clear as it seems. For example, if we take a document on which several authors jointly work, the finalized document is the object of the action. However, during the process of writing, the document can also be used as an instrument to exchange opinions and to find a common understanding. This double character also appears in task management.

Primarily the task management is an instrument that helps individual users to execute their tasks. In this case, it is even obstructive if the task management becomes a goal of its own. The focus should always be placed on the activity to be performed, since this is predominant motive of the actor. On the other hand, the recorded task is an object that is to be used as source of information for other actors. In this sense, the task becomes an object that is to be worked on carefully in order to provide valuable information to other actors.

3.1.1.3 Activity Structure

Regarding the difference between autonomous and collaborative actions there can be a different attitude towards to the collaborating community. In the former case, the community is more or less resolved in favour of an 'objective' relation between the collaborating parties. This means the activities including the actors are transformed into the instruments that help to achieve the goal. In the end, such a relation can be simply replaced by an automatic service with well-defined input, output and performance features. The rules that we can relate to such a process have to be mainly formal.

Turning to collaborative actions the role of the co-workers is better described as that of a temporal community grounded on common rules and practice. Here informal aspects play a much more important role. The joint effort concerning the common work is constantly adjusted and negotiated. The separation of these activities into different tasks does not make sense.

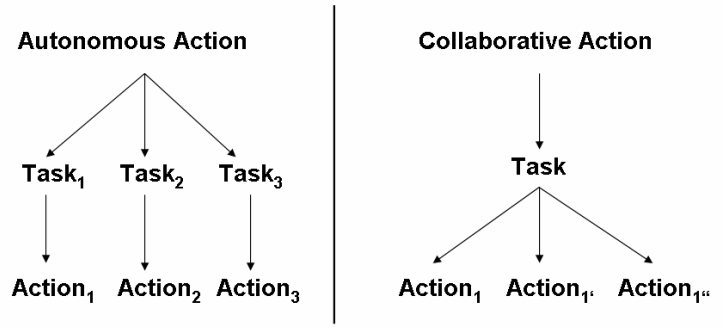

Figure 3.6: Task assignments for autonomous and collaborative actions.

As described in Figure 3.6. Autonomous Actions and Collaborative Actions differ in terms of task assignment. In the former case, every action of a participating subject is represented by a separate task whereas, in the latter case, all actions coalesce into one task and the action become rather similar in character, i.e., their goals are more or less identical.

However, the two types only represent the ends of a spectrum of possibilities; in reality, there are various hybrids. For example, autonomous tasks requires a fixed scheme of interaction which might break down if exceptions occur which require further communication between the participating parties. On the other hand, also Collaborative Actions might have facets to which some participants can better contribute than others. In this case, it might we useful to separate these aspects and describe them in separate tasks. In particular the type of task can change over time, e.g., due to exceptions. The task management must allow for this and provide possibilities to change the mode of collaboration.

Although the common footing in Collaborative Actions is much more pronounced than in Autonomous Tasks, even in the latter case there must be a common understanding on which input and output is expected and which external constraints regarding the process have to be obeyed. This means that even in this case there are certain underlying rules on which the process is based and a community with a common understanding of what is exchanged.

Ways of proceeding are mainly due to experience of the individual user. In communities as well as in organizations novices have to find out how to get their task done. A task management can support the process by providing appropriate information. This information can help users to define certain task and delegate them to other users or to find users and work with them on specific topics or problems.

Concerning the motivation, users are generally driven by multiple goals. On the one hand, they generally like to help other users as well as possible. On the other hand, they usually have so many tasks to accomplish that they have to choose carefully those tasks for which they have enough time. This also concerns the quality of task outcomes. This holds not only the in organisational case but also for communities.

3.1.1.4 Tools and Mediators

Currently only email and calendar systems and a small spectrum of special tools support task handling and give the user an overview of what is to be done. In this respect, emails mainly concern the communicative part of task handling, i.e., the delegation of tasks to other users, although some users also use email systems to store information (cf. D10.1 2006, Section 2.2.2). A central problem in this respect is that email systems are mainly designed to support communication between users but not to track the tasks that are related to this information. Therefore, emails are often stored to preserve the information resulting from a status request. Nevertheless, information is often lost or difficult to find. Relations between emails, even if they belong to connected tasks, are not sufficiently supported.

Similar problem appear in connection with calendar systems which are designed to support the user in planning their time but which are not properly connected to task related communication and information. Although calendar software support often supports the notifications of users regarding planned activities there is no support for tasks that are not assigned to fixed time slots, except for deadlines and reminders. Nevertheless all tasks require a time planning even if there are no time slots reserved for them in the calendar. Moreover, the calendar entries belonging to the same task are not related to that it is difficult to get an overview of all timeslots assigned to a particular task.

Finally, if we consider the main task tools we find that these mainly support users in a rather static way so that task lists become accumulations of unfinished tasks. In particular, functionalities are missing that help users to skip tasks or transfer them to other users. The main tools for this kind of clearance are mainly list on paper or in electronic form. The tidy-up of task items is supported by the assignment of priorities but these are static and do not consider the current situation of the user. For example, it is not considered that a necessary task the deadline of which is approaching immediately is to be handled with higher priority than during the time before.

3.1.1.5 Context Analysis

The context of knowledge workers is mainly given by their inclusion in formal and informal groups in which they exchange information. Synergies between different tasks are often discovered during a conversation in the coffee corner or during a travel. These networks also provide the information about who is to be asked regarding a specific question or to whom a specific task is to be delegated.

Knowledge work is significantly characterised by the occurrence of exceptions that prevent this kind of work from routinisation. In particular these exception cannot be handled in a standard way and require improvisation based on the experience of the knowledge worker. This experience is mainly based on implicit knowledge (Riss et al., 2007). It includes the knowledge of most promising contact persons whom can be asked regarding certain questions or best practice how to proceed in unclear cases. The handling of these cases is often based on analogies to previous cases.

3.1.1.6 Summary

Activity Theory provides a powerful instrument to analyse the situation of knowledge workers, providing constituents to be considered in designing new software. We can base it on a set of core concepts consisting in object-orientation, the principles of internalization/externalization, mediation, three-level schema of activity, and continuous development.

It takes the contradictions in existing work situations into account and helps to understand the individual needs and problems of users. In the previous investigation, we have not applied AT to individual activities but to an entire class of tasks. However, this proceeding is not unusual. For example, Engeström (1987) has applied activity theory to entire paradigms such as scientific work in an abstract sense. This is legitimate and reveals the problems and contradictions that appear on this abstract level. We have applied a schema described by Jonassen and Rohrer-Murphy (1999) to analyse the activities of knowledge workers. This analysis gave us valuable information about the point that is amendable in the current situation.

3.1.2. Coordination theory (CT)

Coordination theory (CT) is an organizational framework for the analysis and improvement of coordination of activities of a number of actors. This concerns a variety of disciplines such as computer science, political science, management science, sociology, organisation theory and others. It centrally addresses the question on how activities of complex systems can be coordinated to improve their interplay.

Malone and Crowston define within CT the term coordination as "managing dependencies between activities" (Malone Crowston 1990 and 1994). CT focuses on the dependencies between activities. Its main claim is that "dependencies and the mechanisms for managing them are general, that is, a given dependency and a mechanism to manage it will be found in a variety of organizational settings" (Crowston et al. 2004). This enables CT to identify dependency types and associate coordination mechanisms. For a concrete dependency, based on the recognition of a dependency type, we may choose and evaluate several alternative coordination mechanisms in order to optimize the management of the dependency, i.e. the coordination process.

3.1.2.1 Fundamental Contributions of Coordination Theory

CT has *three fundamental contributions* according to (Crowston et al. 2004):

First, we have chosen the *definition of coordination,* as given above, out of numerous possibilities in order to facilitate the modelling process. In difference to other definitions, this definition states that dependencies arise "between tasks rather than individuals or units" (Crowston et al. 2004) leading to a simplified modelling of effects for "reassignments of activities to different actors" (Crowston et al. 2004). In the same way, the chosen definition focuses on the "cause for a need to coordinate, rather than on the desired outcome of coordination" (Crowston et al. 2004), which simplifies modelling as well.

Second, a *modelling framework*, i.e., a "theoretical framework for analyzing coordination in complex processes" (Crowston et al. 2004), contributes to user task analysis and modelling.

The main concepts of the framework are tasks, actors and resources. Group action has been analyzed in terms of "*actors performing interdependent tasks*" (Crowston et al. 2004). In this respect, task is considered synonym to activity. "These tasks might require or create *resources* of various types". Thereby, "actors in organizations face coordination problems arising from dependencies that constrain how tasks can be performed."

As a side remark, Malone and Crowston (1994) mentioned that coordination mechanisms rely on "other necessary group functions, such as decision making, communications and development of shared understandings and collective sense-making". Nevertheless, they focus on coordination aspects.

Third, CT presents a *typology of dependencies and coordination mechanisms*. Because of the claim of generality of dependencies and coordination mechanisms, the typology of dependencies assigns for each dependency type and number of possible coordination mechanisms that can be used to manage the dependency. Table 3.1 shows the typology of dependencies and related coordination mechanisms as identified by Malone and Crowston (1994). Several refinements of this typology have been created (Crowston et al. 2004).

Dependency	Examples of coordination processes for managing dependency
Shared resources	"First come/first serve", priority order, budgets, managerial decision, market-like bidding
Task assignments	(same as for "Shared resources")
Producer / consumer relationships	
Prerequisite constraints	Notification, sequencing, tracking
Transfer	Inventory management (e.g., "Just In Time", "Economic Order Quantity")
Usability	Standardization, ask users, participatory design
Design for manufacturability	Concurrent engineering
Simultaneity constraints	Scheduling, synchronization
Task / sub-task	Goal selection, task decomposition

Table 3.1: Examples of common dependencies between Activities and Alternative Coordination processes for managing them (Indentations in the left column indicate more specialized versions of general dependency types) (Malone Crowston 1994).

For example, the dependency 'Task assignments' describes the constraint that a task requires for its execution a certain skill set of an actor. As another example, the producer/consumer dependency describes the situation where one task creates a resource that is created by another task. It is subdivided in three sub-dependencies highlighting sub-aspects, e.g., the precedence sub-dependency represents the fact that actors performing the second task need to be noticed when the required resource becomes available in order to start their task.

A *coordination mechanism* represents the additional work that an actor has to perform to overcome these coordination problems (Crowston et al. 2004). As indicated in Table 3.1, the coordination process of the

dependency 'Task assignments' can be managed by several coordination mechanisms striving all to identify an actor with the required skill set, e.g. by a manager choosing a certain employee.

CT "suggests identifying and studying such common dependencies and their related coordination mechanisms across a wide variety of organizational settings" (Crowston et al. 2004). The identification of coordination mechanisms enables organisations to generate *alternative processes*. This means for the application of CT to a concrete dependency, that the analysis reveals a certain type of dependency and based on this, several coordination mechanisms can be applied and evaluated in order to better manage the dependency, i.e., to improve the coordination.

3.1.2.2 Implications for task management (models)

CT addresses a domain that is highly relevant for task management. Just as CT, task management deals with tasks that are executed by actors with the help of resources. Thereby, dependencies between tasks and tasks, tasks and actors, tasks and resources, resources and resources, and actors and resources can be classified into several dependency types. For example, the already identified dependency types 'Task assignments' and 'Task / sub-task' reside in the direct focus of task management.

The concrete application of CT for task management can take place by transferring the principle of coordination theory – management of dependencies, categorisation of dependencies and generation of alternative coordination processes – to tasks patterns:

- A task pattern contains already explicit representations of dependencies by the relationships to its related resources (i.e. information documents, etc.), tasks (i.e. sub-tasks) and actors (i.e. task executors, task owners).
- A task pattern can incorporate a classification for each of its explicitly stated dependencies. For example, the relationship of the sub-task executor can be classified as a representation of the 'Task assignments' (task - actor) dependency.
- Based on the classification information, several specified coordination mechanisms can be offered and the user then selects one of these. In analogy to CT, the user then can see the possible 'process alternatives' and may choose one of them. This can be represented in the task management application by offering dynamically 'coordination options' that enable the user to choose this coordination mechanism. A representation of the coordination mechanisms based on the dependency types is required as well.
- In terms of the task management model, this requires an attribute for the dependency type.

3.1.3. Object-Oriented Activity Support

This section presents the model 'Object-oriented Activity Support' (OOActSM) as presented by Teege (1996). He developed the model for integrated computer-supported cooperative work (CSCW) systems.

Teege (1996) first develops an activity support model and applies the paradigm of object-orientation to this model. From today's perspective, the presented paradigm of object-orientation is not relevant for activity support; however, the activity support model provides several interesting aspects, which are presented below.

Teege (1996) chose the concept of *'activity' as the central basic abstraction* with the goal in mind to gather a general concept covering all aspects of CSCW. Other possibilities were "workflows, conversations, documents, groups, or conferences".

The *definition of an activity* was targeted for the creation of an integrated CSCW system. This leads to a characterization of activity by three components:

- **Sub-activity structure**: An activity has a hierarchical structure of sub-activities.
- **Executing actor**: An activity is executed by an actor, e.g. a "single person, a group of persons, or a programmed "autonomous agent" in a computer" (Teege 1996)
- **Context**: The context of an activity contains e.g. "tools or information needed, and objects which are changed or produced" Teege 1996).

Teege (1996) states that this definition of an activity covers a "large number of activity kinds", one classification kind consists of "single-user activities without cooperation, cooperative activities by a group, and activities executed automatically by programmed systems" (Teege 1996)

Teege (1996) further introduces two aspects for activities, i.e. *managing activity instances and specifying activity kinds*.

Activity kinds contain a description of how to perform and organize an activity part. In this respect, we can compare activity kinds to the concept of task patterns (Riss et al. 2005)

Teege (1996) reports the observation that within work processes often a design aspect is involved, i.e. an actor not only executes the work but as well finds alternative ways to perform and organize parts of the activity. Therefore, the proposed model first supports the specification of activity kinds and second supports their design. Teege (1996) defined three requirements for the design of 'activity kinds':

- The "specification of activity kinds must not be restricted to a single specification mechanism" (Teege 1996). Teege mentions a control flow specification as an example which may be useful for some but not necessarily for all activity kinds.
- The "model must support a wide spectrum of degree of detail" (Teege 1996). The specification of activity kinds may be rather general when details about the activity are not known, e.g. a single person conducts an activity. On the other hand, an activity kind specification can contain a detailed description and individual steps.
- It should be possible to "reuse existing specifications by extending or modifying them" (Teege 1996). Teege mentions the example situation of sending a letter where the sub-activities depend on the used medium, e.g. mail or electronic mail.

Activity instances are the entities that "allow the representation and support of concrete activities" (Teege 1996). Teege (1996) defines three

types of support by the system for activity instances defined in the model.

- **Structuring**: Structuring contains the maintenance of the activity structure, e.g. the sub-activities.
- **History**: The history collects information on the activity, which is "not part of the activity kind" (Teege 1996) and which is used to determine on how to proceed with the activity. In the same way, after the completed execution of an activity, we can use the history to "design activity kinds for similar activities, thus reusing the experience gained while executing the activity" (Teege 1996)
- **Execution**: Executing sub-steps support the execution of activities by organizing the "sequencing of steps or the coordination among different actors". Sub-step execution is only supported in case that the computer performs the respective sub-step, e.g., sending an email. Step sequence organization requires a mechanically interpretable execution scheme for each activity, e.g., like in workflow applications

To enable the support as described in the paragraph above, Teege (1996) defined several requirements. Thus, the representation of all activity parts requires rich structuring facilities, the maintenance of a state for activity instances and activity instances should provide storing facilities for history information and for activity-specific execution procedures.

Moreover, Teege (1996) states that the *model explicitly doesn't include specific theories of coordination or communication* in order to enable coexisting theories in the model. This is due to the reason that each of these can be "used for modeling corresponding specific activity kinds" [Teege (1996)]. Teege (1996) mentions explicitly "activity theory (Kuutti, 1991), coordination theory (Malone and Crowston, 1990), communication for action (Winograd, 1988), or the action workflow approach (Medina—Mora et al. 1992)".

Summed up, OOActSM provides several *contributions for task modelling* having in mind that the analogy of activities to tasks is obvious:

- Choosing the concept of 'activity' as the central basic abstraction over the other possibilities like "workflows, conversations, documents, groups, or conferences".
- Separation between activity instances and activity kinds: Work processes involve often a design aspect, which is captured by activity kinds, whereby activity instances represent the representation and support of concrete activities.

3.2. Exemplary Task Model Role Models

Having treated general task modelling methodologies in the last section, we now focus on concrete task models from literature, products and research prototypes. The goal is to describe good examples of task modelling that serve as role models for the Nepomuk task modelling. We also refer to these models and partial aspects as *how-concepts* since they throw some light on *how* we can describe the previously introduced concepts in a formal way.

In particular, this section presents exemplarily concepts on how to describe model tasks in a concrete way. First, the analysis includes research prototypes such Unified Activity Management (UAM) and Task

Manager (1993). Second, selected modelling aspects, such as tasks vs. discrete sub-tasks and task states, are highlighted based on scientific literature.

3.2.1. Unified Activity Management (UAM)

IBM's Unified Activity Management (UAM) (Moran et al. 2005) aims at building a generic computational construct to represent human collaborative activities by capturing the salient elements of the idea of an activity. Additionally, the activity construct should be amenable to support the scaffolding of infrastructure and tools around the concept of the activity. An activity describes the people involved and their roles, the resources used (including tools, people and activity artefacts), the results produced, the events related to the activity and the relationship to other activities.

Activities seldom exist in isolation. Rather, they are related to other activities (see Figure 3.7). The main relationship is that of sub- and super-activity whereby an activity can be decomposed into one or more sub-activities and is itself part of one or more super-activities. Furthermore, an activity can depend on (as a consumer of) other activities (as producer).

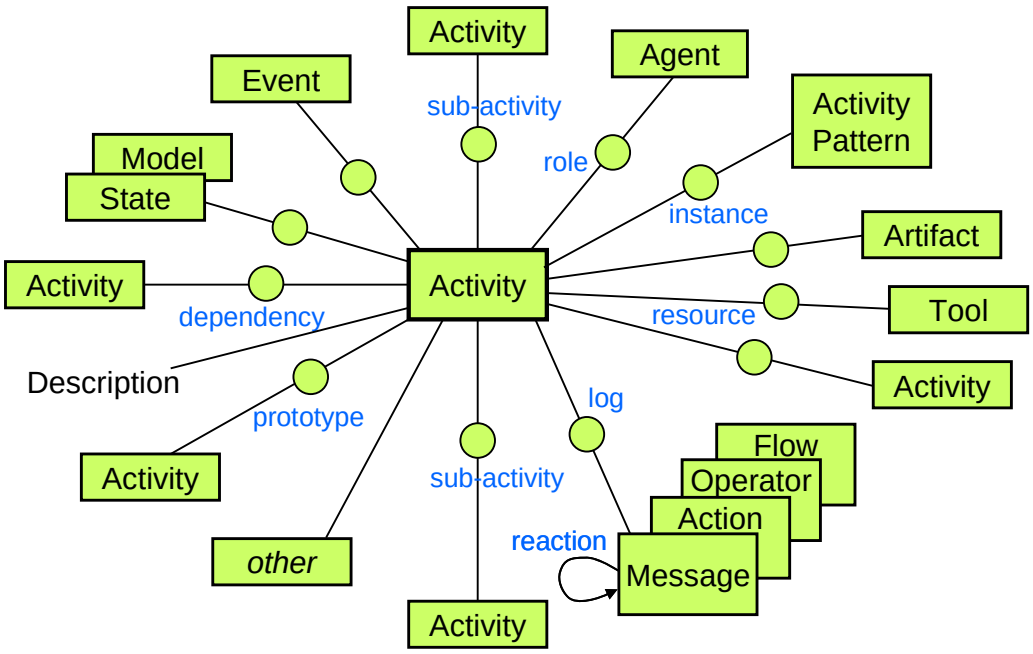

Figure 3.7: Unified Activity Representation (a "Metamodel for Work") (Moran 2005a)

The objectives of the UAM activity model (Moran et al. 2005) include:

- **To organise work around activities instead of tools and artefacts** – the activity description contains pointers to resources related to the activity. That is, activity descriptions encode metadata that bind the activity artefacts together. To this end, Moran et al. (2005) are developing a core Unified Activity ontology based on OWL.
- **To guide, support, and coordinate work but not to overly constraint it** – people are free to adapt activity descriptions to the work situations. Furthermore, Moran et al. (2006) are planning to extend UAM with access control policies and

constraint handling to support interoperability with formal processes such as workflows.

- **To provide a single place for people to manage the whole of their activities** – having one place to organise all (shared) activities provides a higher-level view for reflection including planning, prioritisation and negotiation. Furthermore, the core ontology defines a generic collaboration activity to unify activity-related information across applications.
- **To capture, reuse and evolve best practices in activity patterns** – activity patterns can be developed and incrementally refined by analysing variations of activity instances.
- **To integrate informal business activities and workflow-driven business processes** – activities complement workflow processes by delegating complex social activity to people via UAM. Figure 3.8 describes the relation between processes, activities and actions (well-defined pieces of work e.g. write email or browse website). Whereas actions are spontaneous and reactive and processes are fully pre-planned and directed, activities provide a means for people to be reflective about their work (Moran 2005).

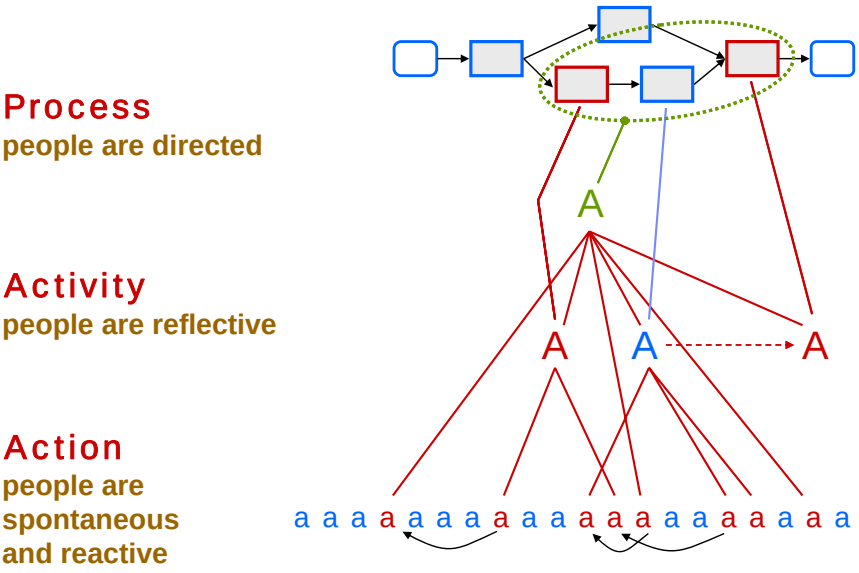

Figure 3.8: Modes of working (Moran 2005)

Activity instances can be formally and incrementally refined into best practice in the form of activity patterns from which other activity instances emanate (see Figure 3.9). This evolutionary process involves analyzing variations between related instances. However, UAM does not define any methodology nor does it provide any user guidance for abstracting selected parts of the task description. Furthermore, the clear conceptual separation of processes and activities in UAM make it less amenable to formalize activity patterns as process fragments, which can be readily orchestrated in a business-driven workflow process. On the other hand, the Nepomuk task management model does not make such a clear distinction. This leaves additional research avenues open to exploration in subsequent work.

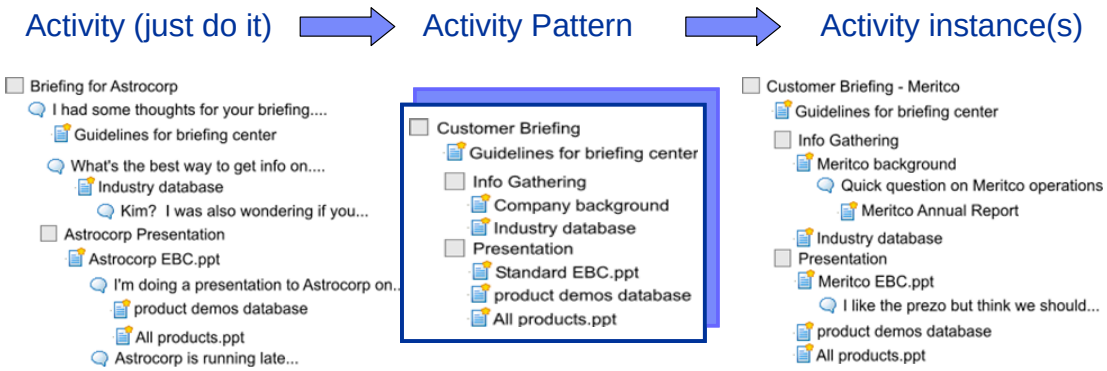

Figure 3.9: Activity pattern lifecycle (Moran 2005a)

Figure 3.10 shows an Eclipse-based prototype of UAM developed at IBM. The Unified Activity ontology, however, is at present incomplete and not yet generally available.

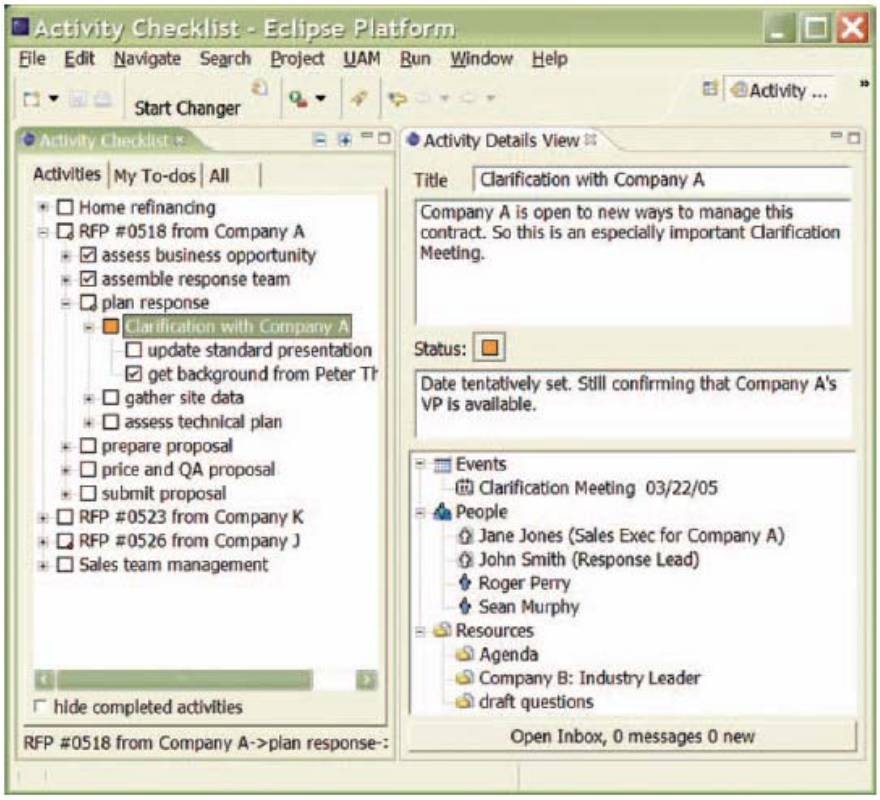

Figure 3.10: Shared activity checklist in an Eclipse-based prototype (Moran et al. 2005)

3.2.2. Task Manager (1993)

Task Manager (Kreifelts et al. 1993) is a software system for sharing to-do lists. This includes the sharing and distributed manipulation of a set of common tasks.

Task Manager highlights already in 1993 several highly relevant aspects. The sections below present selected aspects being relevant for the modelling tasks.

Guareis de Farias et al. (2000) reconstructed the main objects of the underlying "cooperative model" of Task Manager in a class diagram as presented in Figure 3.11.

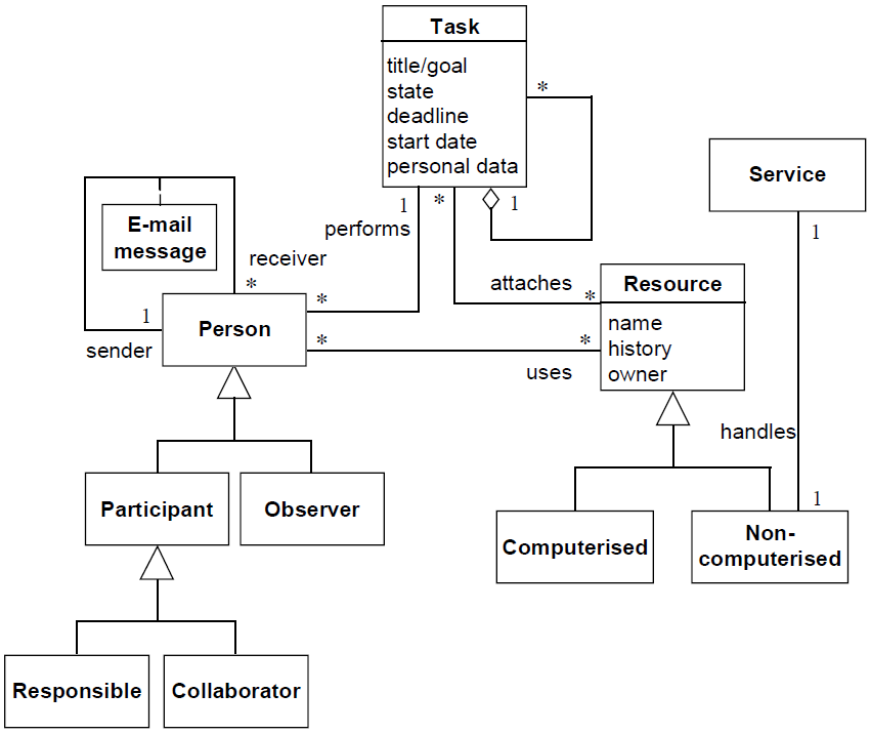

Figure 3.11: Task Manager class diagram (Guareis de Farias et al. 2000)

Task manager is a tool, however it incorporates a "cooperative model" being employed in the development of Task Manager (Guareis de Farias et al. 2000).

3.2.2.1 Addressed Issues

Task Manager focuses on resolving identified problems of 'office procedure systems', today's business process management systems. This includes the "rigidity of pre-defined procedures" and "isolation from informal communication and information sharing" based on observations conducted on previous office procedure systems (Kreifelts et al. 1993). "Rigidity of pre-defined procedures" corresponds in today's language to the **rigidity of process models** in **business process management software**. Kreifelts and his co-authors argue for avoiding the implications of rigidity for "treating models of cooperative work as resources to be defined, modified, and referred to for information purposes instead of as prescriptions to be adhered to" (Kreifelts et al. 1993). The other problem domain of existing office procedure systems' **isolation from informal communication and information sharing** is today still relevant for business process management software. This is relevant despite research efforts for this software category and for related categories such as groupware and collaboration support systems. Kreifelts et al. (1993) argue to overcome the isolation issue that "future coordination support systems have to be able to interface to existing computer systems that support the actual work".

3.2.2.2 Task model entities

Task Manager has several task model entities, referred to as components within (Kreifelts et al. 1993).

3.2.2.2.1 Task - Result- vs. Procedure- vs. Information-Sharing-Orientation

Kreifelts et al. (1993) identify for Task Manager several aspects of a task depending on the predominant orientation of the task:

- A result-oriented task can be regarded as "a project, i.e. a common goal of a set of people" (Kreifelts et al. 1993).
- A procedure-oriented task consists of a task-breakdown into several sub-tasks and corresponding dependencies between the tasks and associated documents. They note that "the more detailed specifications are given, the more a task resembles an office procedure with causal dependencies between sub-tasks and documents of a task" (Kreifelts et al. 1993).
- A information-sharing-oriented task represents a "simple folder with little or no structure defined" where a task is "simply a shared container of sub-tasks, documents and/or services, and messages that people exchange about in a common task" (Kreifelts et al. 1993).

3.2.2.2.2 Documents / Services

Kreifelts et al. (1993) state that resources are needed in order to achieve the goal of task. They attach **resources** to tasks and define them as **"pointers" to various kinds of computerized objects**. Resources include documents as well as "rooms, budgets, machinery, etc." (Kreifelts et al. 1993). For Task Manager, services handle the resources outside the task manager and refer to them from the Task Manager.

3.2.2.2.3 People / Users

Kreifelts et al. (1993) distinguish for the involved resource 'person' several "levels of participation and of competence".

- *Participants* are involved people that all have within the task "equal access rights to the attributes of a task, its documents and services and its messages" (Kreifelts et al. 1993).
- Upon invitation, other people can take part in a task by either being participants or *observers*. "Observers are people interested in the completion of the task with read access only to any information and the right to participate in the informal message exchange associated with the task" (Kreifelts et al. 1993).
- The role *person responsible* assigns the responsibility for a task to a dedicated person. This includes "exclusive write access to some of the tasks attributes, e.g. state, start date and deadline, and only s/he may reassign the responsibility of the task to another person" (Kreifelts et al. 1993).

3.2.2.3 Task Model Attributes

Kreifelts et al. (1993) described attributes that further specify the task model entities. Table 3.2 lists several of their core task attributes:

Attribute	Description
Title of a task (the only mandatory attribute)	"both identifies a task to its participants and gives a short and concise description of its goal"
Deadline of a task	The "system reminds the user of approaching deadlines, but does not enforce any actions with respect to overdue tasks."
Task state	See table below

Table 3.2: Core task attributes (Kreifelts et al. 1993).

The 'person responsible' has special rights on a task, e.g. she may set the task state, as shown in detail in Table 3.3:

Task state	Possible task state values
Completion state	Not finished / finished
Pending task	"Pending, i.e. there is a causal dependence on another task not yet finished, or not pending."
Task acknowledgement	"Task can be acknowledged by the responsible actor. This is to inform the co-workers of the responsible person's awareness and acceptance of the task s/he has been assigned to."

Table 3.3: Task states (Kreifelts et al. 1993).

Several attributes detail how the task should be performed, see Table 3.4:

Attribute	Description
Time-related data	E.g. start date
Data that describes causal dependencies between tasks	
Data that describes causal dependencies between tasks and documents	
Personal data attached to a task, such as notes etc.	

Table 3.4: Task performance detail attributes (Kreifelts et al. 1993).

The dependencies, e.g., between tasks, are described locally, i.e., as user specific. Kreifelts et al. (1993) describe these rather short and ambiguously by referring to latter attributes: "The latter attributes are purely local and are not distributed to and shared by the other participants." (Kreifelts et al. 1993). Prinz (1994) talks about the same aspect with respect to a very similar model for the Computer Supported Cooperative Work (CSCW) prototype TOSCA: "Resources can be associated to each task, such as documents, forms, calendars, etc. This is done by appropriate **relationship** objects. These are **described user specific**, so that each user gets individual information about the people who are responsible or the forms which are valid for him." (Prinz 1994)

"Documents and/or services may be attached to any task in which the user participates at any time." We have attached some of the respective attributes in Table 3.5:

Attribute	Description
"Name	
History of who did what and when	
The owner of the document"	
Abstract	The Abstract "contains an informal text description of the document and it frees the user of having to transfer, open and read the entire document when s/he is only interested in a resume"

Table 3.5: Document attributes (Kreifelts et al. 1993).

3.2.2.4 Operations

The following Table 3.6 lists several operations and the allowed entities that are allowed to execute these operations as presented by Kreifelts et al. (1993):

Who?	What?
Users	Create tasks and sub-tasks
	Create dependencies between tasks
	Create dependencies between tasks and documents
	Set and modify attributes
	Add, modify, and remove documents and service requests
	Add, modify, and remove documents and service requests
User (Person responsible)	Refuse responsibility and reassign it to another user
User (Any participant)	Introduce new participants or observers to a task
Tasks	Copy and paste or move around freely
System	Distribute information on tasks
	Makes available resources across the (world-wide) network
	Keeps the data up-to-date
	Resolves conflicts of synchronization
	Each user has instant access to the shared tasks s/he is involved in
	Guarantees a consistent view on tasks for each participant
	Keeps track of the actions the users take
	Monitoring and task tracking at execution time
	Report generation after completion of a task is rendered possible

Table 3.6: Operations of Task Manager (Kreifelts et al. 1993).

3.2.3. Further selected aspects

This section presents modelling aspects, i.e. tasks vs. discrete sub-tasks and task states, having been identified as subject to discussion.

3.2.3.1 Unified View on Tasks vs. Discrete Sub-tasks – FRODO Example

Thinking about the structure of decomposed tasks having sub- and super-tasks, leads to the question how we can model these tasks and sub-/super-tasks. In this respect, the question arises, whether a unified view on tasks is reasonable, e.g., by integrating the representation of tasks and sub-tasks in a single data structure. Otherwise, we would need a discrete data structures for both tasks and sub-tasks.

A rationale for following a unified view on the task structure is presented by means of the example of FRODO. FRODO is a research project and deals with "methods and tools for building and maintaining distributed Organizational Memories in a real-world enterprise environment" (FRODO 2005).

Thereby, FRODO aims at knowledge-intensive activities with a process-oriented knowledge management approach by using the concept of weakly-structured workflows. FRODO TaskMan demonstrates weakly structured workflows being an "agent-based workflow management system integrated in organizational memory information systems" (FRODO 2005).

With respect to the unified view on tasks in FRODO, the following design rationale underlies the FRODO TaskMan weakly-structured workflow-system (cf. FRODO 2003):

- complete flexibility of the workflow execution; modelling and execution is intertwined
- lazy/late-modelling
- hierarchical decomposition of workflow activities
- agent-based architecture: TaskInstance-Agent is responsible for executing the task respectively "getting the task done"

Due to that, FRODO decided to use the concept of "Task" as the only object for representing workflows and activities. A task can possess multiple sub-tasks and at most one super-task. A task without a super-task is a root task and called 'workflow'.

The task has all ingredients for being a full member of a control and data flow of a workflow, i.e., it has pre- and post-conditions, input/output-container, and links to other relevant tasks (e.g., sub-tasks). (Especially, in FRODO there is no separate control or data flow, all of it is included in the tasks and built during start-up of the agent system)

Reasons for this are:

- a task is a self-containing entity, describing everything what is needed to be executed, among others, pointers to super-task, to workflow (i.e., root-task), to predecessors respectively start-conditions, all sub-tasks
- flexibility in changing tasks on-the-fly, no need to transform a workflow in a task or a task into a sub-task-object as well as take a sub-task out of a workflow and making it a workflow (or workflow model) (i.e., root task) by itself.

- ease of handling in the FRODO agent system: distribution of tasks to agents including required relations (self-containing task descriptions)

To conclude, following a unified view on the task structure has benefits that outweigh the reasons for a separate task and sub-/super-task structures.

3.2.3.2 Task states

Task states describe the operational status of a given task at any instant, and may be set explicitly by the user or automatically determined by the system.

Grebner (2006) proposed five operational task states for each task that are depicted in Figure 3.12. The state of a sub-task may vary independently of its super-task. The status value is derived from the status value of the latest operation on the task. While the original intent is to associate task states to task *services* representing task instances within the service-oriented task management paradigm rather than human-centric tasks, the simplicity of Grebner's task state model is particularly appealing.

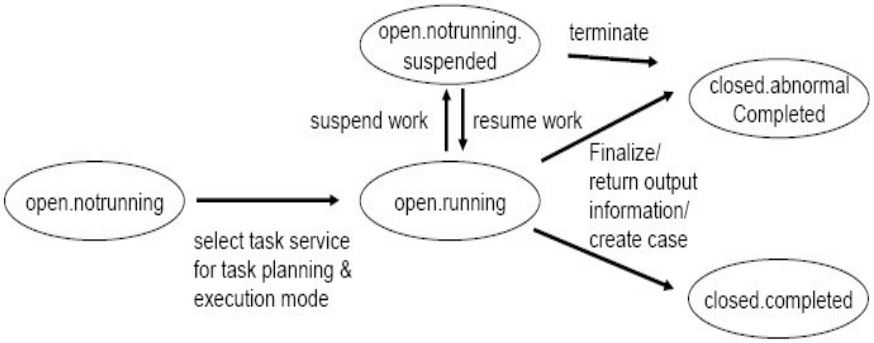

Figure 3.12: Overview of operational task states (Grebner 2006).

Operational task state	Description
open. notrunning	The task with the invoked operation has not been selected in the task inbox yet, i.e. the user does not work on it.
open. notrunning.suspended	The user already begun working on the task operation, but the operation has been suspended, i.e. the work on the operation rests.
open. running	The task with the marked operation has been selected from the task inbox and the user works on it.
closed. completed	The task operation has been completed successfully, i.e. it is finalized and the results are delivered.
closed. abnormalCompleted	The task operation has produced some errors during execution, i.e. it is finalized and the results are delivered, but these results may not be correct. The exact reason, i.e. the exception, is submitted separately with the output information.

Table 3.7: Operational task states (Grebner 2006).

Dourish et al. (1996) proposed the use of *constraints* to model relationships between tasks, and consequently their respective task

states. Instead of (procedural) *transitions* between task states, constraints specify ongoing (declarative) *relationships* between these states as the tasks are operated on, e.g., "document formatting should not be performed unless the text has been approved" (Dourish et al. 1996). This approach suggests that it is useful to distinguish *temporal* (ordering of tasks) and *dependency* relationships.

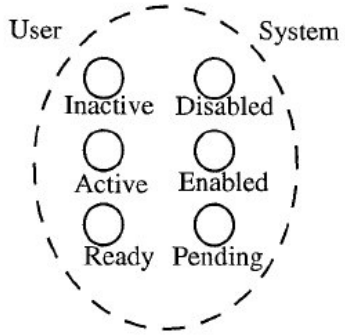

Figure 3.13: Overview of Freeflow task states (Dourish et al. 1996).

Context	Task state	Description
User	Inactive	Before any work has been done
	Active	User has begun work
	Ready	Work is complete
System	Disabled	A task on which the task depends has not been completed
	Enabled	A task whose preconditions for starting have been met
	Pending	A task prevented from completing due to *external* dependencies

Table 3.8: Freeflow task states (Dourish et al. 1996).

Caramba (Dustdar 2004) supports the continuum of process types from ad hoc processes with no underlying process models to modelled processes and combinations of the two. However, its main emphasis is on the ad hoc processes. The system allows the user to start from a process template and subsequently to deviate by omitting activities from the template or by adding new ones.

When a Task is instantiated, it becomes an Activity[1] in the initial state *new*. It remains in this state until it is *read* by the user. Subsequently, the user may choose to *suspend* the activity or to start work on it (*active*). However, the user may also choose to *delegate* the activity to another user which transfers the responsibility for the activity to the recipient. On the other hand, other users could be kept informed, as an observer of the activity, by *forwarding* the activity to them; the sender remains responsible for the activity. As applications associated with the activity are invoked, the activity is said to be *applied* which, upon completion, is marked as *done* which automatically triggers coordination

[1] Or *instantiated tasks* – not to be confused with activities from Activity Theory.

with any dependent activities. Caramba further allows activities, which are no longer required to be *archived*. In the event that no executor can be identified, the *no-route* state is triggered.

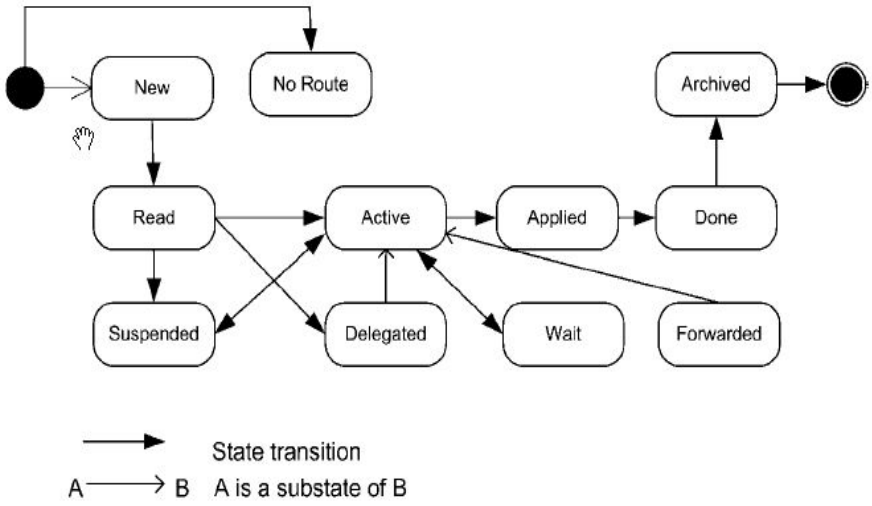

Figure 3.14: Overview of Caramba task (activity) states (Dustdar 2004).

FRODO TaskMan (DFKI 2003) supports the modelling and enactment of weak workflows as the context for information support in knowledge-intensive work situations. The worklist handler presents the relevant tasks and their task states in addition to supporting dynamic process modification and integrated information access. This model presents one concern: The flexibility of the numerous transitions to and from each task state could pose usability problems when the user is confronted with a gamut of choices he could make at any instant. A fewer number of transitions is, in general, preferable from a usability perspective.

Task state	Description
Initiated	A task instance starts here and waits for all preconditions to be met before proceeding
Processible	The task is ready to be processed
In Progress	A user is executing the task
Active	The task is processible but is temporarily paused
Suspended	The task is suspended indefinitely
Terminated	The suspended task is terminated and subsequently deleted
Completed	The task is completed

Table 3.9: FRODO TaskMan task states (DFKI 2003).

4. Task Management Model Requirements – Scoping

This section derives and defines the scope for a task management model based on the requirements from the Nepomuk case study scenarios.

The task model presented in this document has to fulfil a dedicated purpose. This purpose is determined by the case studies conducted in the work packages WP8000 to WP11000 of the Nepomuk project. These packages define the application domain of the Nepomuk project and thereby are eligible to define the requirements needed in order to address the intended application domain.

First, the task management-related requirements from the case study scenarios are collected in order to give an overview on the specific requirements and the surrounding setting. Second, these requirements are integrated into a coherent view.

4.1. Requirements from the Nepomuk case studies

This section collects the task management-related requirements from the case study deliverables.

In the following, the requirements regarding task management are shortly mentioned. Additionally, the goal of the case study and its related scenarios are mentioned briefly in order to provide an overview on each case study.

4.1.1. Task Management functional requirements in WP8000

WP8000 focuses on research processes in the bioscience company Institute Pasteur. Therein, laboratory work plays an important role.

WP8000 considers task management mainly in the context of project management. For example, the analysis of the top-level user needs leads to the use of task management in the project management context for "efficient coordination, planning, and reliable implementation of a preset sequence of hierarchical tasks, e.g., protocol implementation" (Nepomuk D8.1 2006, section 2.3).

In a conducted questionnaire for examining user needs, task management functionality, i.e. "coordination, planning, and implementation" (Nepomuk D8.1 2006, section 2.3) is ranked rather weak in comparison to other user needs such as preservation of all data and information clarity.

Based on the user needs analysis, to-be-scenarios and further evaluations, WP8000 derives **functional requirements**. Table 4.1 cites the requirements that WP8000 considers relevant for WP3 Task Management. The requirements are grouped in functional areas.

Functional area	Requirement identifier and name	Requirement summary
FA-01: Entry and tagging	REQ-01-01: Semantic tagging of files, web pages, and emails	Assigning meta-data to an object, either restricted by a pre-existing domain ontology or open for creation of ad-hoc properties.
	REQ-01-02: Semantic tagging	Assigning meta-data to words, phrases, or document sections, either

	of phrases inside documents	restricted by a pre-existing domain ontology or open for creation of ad-hoc properties
	REQ-01-03: Semi-automatic tagging	A) Automatically extracting and formalizing meta-data from the available unambiguously structured information B) Rules-based learning algorithms for making tagging suggestions and learning from the user feedback
FA-03: Sharing	REQ-03-02: Automatic notification	To automatically notify and be notified of relevant changes in the shared information (e.g., a new critical problem has been encountered, an important result has been achieved, a relevant experiment has been conducted).
FA-05: Representation, Visualization, and Analysis	REQ-05-02: Report draft generation	Based on content's semantic structure, automatically generate drafts of documents from other documents, e.g., generation of publication or project report drafts from research notes.
	REQ-05-03: Workflow management	Tracking of tasks, experiments, projects, people, and the relationships between them
	REQ-05-04: Compatibility with mobile devices	Availability of information on mobile devices such as a Palm pilot or a mobile phone (both for reading and modification)
FA-06: Protection of intellectual property	REQ-06-02: Audit trail	For each modification on the tag level, it is needed to know who modified it, and when. This is important for collaborative editing and protection of the scientific intellectual property.

Table 4.1: WP8000 functional requirements & areas (Nepomuk D8.1 2006).

4.1.2. Task Management functional requirements in WP9000

WP9000 focuses on a professional business services company scenario by means of the example TMI, an international network of consulting companies. The deliverable D9.1 addresses the "analysis on the professional business services domain in order to derive general domain requirements and elicit concrete user requirements from TMI" (Nepomuk D9.1 2006).

WP9000 considers as well task management mainly in the context of project management. D9.1 depicts in several use cases, i.e. to-be-scenarios, the use for a task-project management: (TMI use case 1 – **Sales**) Nepomuk D9.1 2006, section 4.4.1, (TMI use case 2 – **Standard Product Development**) Nepomuk D9.1 2006, section 4.4.2, and (TMI use case 3 – **Customised Product Development, Sharing and Update**) Nepomuk D9.1 2006, section 4.4.3. They give us some guidance by showing steps to pursue for the creation of a document in a sales meeting preparation, in the creation of a new standard product and in the customization of a product.

From these use cases the requirement DOM-01-09 for 'Semantically enhanced Task-Project Management' is derived, see Table 4.2.

Functional area	Requirement identifier and name	Requirement summary
FA 01: Desktop Layer (Individuals - PIM)	DOM-01-09: Semantically enhanced Task-Project Management	Integrated task-project management support for the individual knowledge worker in a networked environment. The support for individual task-project management must be embedded in the user's personal desktop and has to tackle:
		- ad-hoc task planning and flexible changes,
		- collaborative work on task,
		- knowledge-intensive tasks with a huge amount of personal as well as group information objects, and
		- integration into organizational processes

Table 4.2: WP9000 functional requirements & areas (Nepomuk D9.1 2006).

4.1.3. Task Management functional requirements in WP10000

WP10000 focuses on the research organisation of SAP AG, i.e. SAP Research. WP10000 analyses in the deliverable D10.1 (Nepomuk D10.1 2006) the "demands that result from the software research and development process at SAP and generally in large organisations".

WP10000 considers task management as a main concept for dealing with the inherent complexity in this process by giving the participants tools at hand that enable them to "accomplish their individual tasks as part of the whole" (Nepomuk D10.1 2006).

WP10000 defines several use cases with task management involvement in the research domain. These use cases depict the to-be-situation with applied Nepomuk system and in particular a task management system. These use cases have been developed based on the combination of ("to-be") personas related scenarios and a situation description of the SAP Research process. In concrete, the use cases consist of scientific paper creation, proposal creation, project work, project management and transfer projects for scientific results.

Based on these use cases, functional requirements are derived for the Nepomuk system. Table 4.3 shows the requirements that affect task management.

Functional area	Requirement identifier and name	Requirement summary
FA 01: Data-centric Requirements at Individual Level	DOM-01-03: Get support dealing with emails	The system provides support to users structuring their email inbox as well as the creation of new emails.
FA 02: Activity-centred Requirements at Individual Level (TM)	DOM-02-0: Core Task Management	The execution of tasks is supported by a task management that helps to distribute the work among users in the network and that fosters the communication between them.
	DOM-02-02: Calendar support for tasks	The execution of tasks requires periods of time in which this can take place. The corresponding planning is

		supported by connecting task and calendar functionalities.
	DOM-02-03: Support for specific task types	The execution of specific tasks which are regularly repeated can be supported by the provision of templates that describe a pattern according to which a task can be executed to guide the user.
	DOM-02-04: Offline support for tasks	If the users have no access to the task management system, e.g., on meetings or travels, it is necessary to has offline access to the task information.
	DOM-02-05: Time Management	The available time of users plays a central role for the delegation of work. The system checks whether a user has enough time to work on a task.
	DOM-02-06: Task Tracking	For the execution of tasks in time it is necessary that the supervisor of a task gets enough information about the status of delegated work.
FA 03: Social Networking Requirements	DOM-03-02: Compare structured information	Users can compare the structures related to certain types of information to find overlaps. The aim is to find a common structure for both parties.
	DOM-03-03: Collaborative Tasks	Some tasks cannot be executed by individual users but require the cooperation of several co-workers who work together. This cooperation is supported by collaborative tasks.

Table 4.3: WP10000 functional requirements & areas (Nepomuk D10.1 2006).

4.1.4. Task Management functional requirements in WP11000

WP11000 focuses on the application of Nepomuk at the open-source on-line community of Mandriva Linux users in order to equip community members with a "new generation tool for sharing knowledge related to the open-source Mandriva Linux project" (Nepomuk D11.1 2006).

WP11000 considers task management in the context of collaboration support. In the scenario set "a social semantic help desk at work" (Nepomuk D11.1 2006), task management supports the scenario "A group of experts write collaboratively a manual on virtualization". The group can break down the work by defining "specific and general activities", i.e. tasks. Thereby, task patterns can be used in order to re-use tasks that have been conducted by other groups during the writing of a manual.

This leads to the definition of several requirements regarding task management, see Table 4.4:

Functional area	Requirement identifier and name	Requirement summary
Knowledge work process support	Personal workflow support	A service should provide a graphical assistant that lets the users define automated tasks he wants to run on this computer. These tasks should be fired either when a given event occurs (example: an expert with expertise in the area of webcams

		installation on Linux is now available online for real-time help) or periodically. The corresponding workflow to be triggered when the event occurs should be defined graphically by the user as well.
	Task pattern support	Users should be able to define general task patterns for a given activity and to store them in the system for future use by them or others. Example: in the scenario above related to the collaborative writing of a manual, the involved users will be able to instantiate a "task pattern" describing their activity for coordinating their work, distribute the various tasks and continuously assess the progress toward the objective.
	Notification requirements	Users should be able to express the notification they want to receive using advanced rules. Notification rules should support both content related events and user related events. Example: content updates, content creation, or user appearing on the network.
	Contextual recommendation	While writing some document, the system should propose live related resources (documents or persons) that may be of interest to the user in the context of his current activity. Example: while an expert answers a question, he may request assistance from the system for getting directly while typing some relevant resources he will point the reader to.

Table 4.4: WP11000 functional requirements & areas (Nepomuk D11.1 2006).

4.2. Consolidated task management requirements

This section integrates the collected requirements from the Nepomuk case study deliverables into a coherent view. This coherent view defines the requirements, on which section 5 defines the task management model.

Table 4.5 shows the consolidated functional task management requirements for core task management functions. Core functions are the functions that are considered relevant for a first implementation level.

Req-#	Requirement	Requirement details
1	Task Creation	Every user can **create new tasks**
2	Task Planning	**Task decomposition** – Define sub-tasks **Sub-task dependencies** – Create relationships between sub-tasks within a task **Define input and output** of tasks
3		Ad-hoc task planning and **flexible changes** ([WP9000] DOM-01-09)
4		Support users **in structuring their email inbox** as well as the creation of new emails. ([WP10000] DOM-01-03)
5	Task execution support	The **transfer** of tasks as well as the **invitation** to tasks requires a negotiation between the requesting party and the addressee. ([WP10000] DOM-02-0)
6		Integrate personal as well as group **information objects** ([WP9000] DOM-01-09)

7	Task Patterns	**Task patterns** as the central medium to distribute work experience on tasks Support for the execution of specific, regularly repeated tasks by provision of templates that describe a pattern according to which a task can be executed to guide the user. ([WP10000] DOM-02-03) and ([WP11000] Task pattern support)
8		- **Task pattern creation** should emanate from existing cases. - Leverage an abstraction process (removal or generalization of context dependent and personal parts of the task description)
9		**Task pattern update** works according to similar principles as for the creation, i.e., - In case that a user deviated from the pattern during the task execution, the user is encouraged to provide information in an unobtrusive way, supported by the Nepomuk task management
10	Time Management	Ensure that (at least the important) tasks are executed in a proper manner. - Efficient **prioritization of tasks** - Ensure that users have enough time for their tasks Assumption: Knowledge workers usually have more tasks than they can actually accomplish.
11		**Calendar support** for tasks: The execution of tasks requires periods of time in which this can take place. The corresponding planning is supported by connecting task and calendar functionalities. The available time of users plays a central role for the delegation of work. The system checks whether a user has enough time to work on a task. ([WP10000] DOM-02-02) and ([WP10000] DOM-02-05)
12	Task Tracking	**Tracking of tasks**, experiments, projects, people, and the relationships between them ([WP8000] REQ-05-03)
13		For the execution of tasks in time it is necessary that the supervisor of a task gets enough information about the **status of delegated work**. ([WP10000] DOM-02-06)
14	Task History / Audit Trail	For each modification on the tag level, it is needed to track the **audit trail of the changes** e.g. who modified it, and when. This is important for collaborative editing and protection of the scientific intellectual property. ([WP8000] REQ-06-02)
15	Offline support for tasks	Make task information **accessible offline** in case of no online access to the task management system. Example: On meetings or travels. ([WP10000] DOM-02-04)

Table 4.5: Consolidated functional task management requirements: Core functions.

Table 4.6 shows the consolidated functional task management requirements for some extended functions that are not considered as the core functions of the Nepomuk task management.

Req-#	Requirement	Requirement details
16	Task access	Availability of information on **mobile devices** such as a Palm pilot or a mobile phone (both for reading and modification) ([WP8000] REQ-05-04)
17		Support for individual task-project management is embedded in the user's **personal desktop** ([WP9000] DOM-01-09) **Relevant work packages:** WP1000
18	Notification	**Automatic notification**: - Automatically notify and be notified upon relevant changes in the shared information -Example: a new critical problem has been encountered, an important result has been achieved, a relevant experiment has been conducted ([WP8000] REQ-03-02) **Relevant work packages**: WP2000
19		**Notification rules**: - Users should be able to express the notification they want to receive using advanced rules. - Notification rules should support both content related events and user related events. - Example: content updates, content creation, or user appearing on the network. ([WP11000] Notification requirements)
20	Contextual recommendation	While writing some document, the system should **propose live related resources** (documents or persons) that may be of interest to the user in the context of his current activity. Example: while an expert answers a question, he may request assistance from the system for getting directly while typing some relevant resources he will point the reader to. ([WP11000] Contextual recommendation) **Relevant work packages**: WP2000
21	Content provisioning and comparison	**Report draft generation**: Based on content's semantic structure, automatically generate drafts of documents from other documents, e.g., generation of publication or project report drafts from research notes. ([WP8000] REQ-05-02)
22		**Compare structured information**: Users can compare the structures related to certain types of information to find overlaps. The aim is to find a common structure for both parties. ([WP10000] DOM-03-02)
23	Tagging	**Semantic tagging** of files, web pages, and emails: Assign meta-data to an object, either restricted by a pre-existing domain ontology or open for creation of ad-hoc properties. ([WP8000] REQ-01-01) **Relevant work packages**: WP1000, WP2000
24		**Semantic tagging** of phrases *inside* documents: Assign meta-data to words, phrases, or document sections, either restricted by a pre-existing domain ontology or open for creation of ad-hoc properties ([WP8000] REQ-01-02) **Relevant work packages**: WP1000

25		**Semi-automatic tagging**: A) Automatically extract and formalize meta-data from the available, unambiguously structured information B) Rule-based learning algorithms for making tagging suggestions and learning from the user feedback ([WP8000] REQ-01-03) **Relevant work packages**: WP1000, WP2000
26	Personal workflow support	Provide a **graphical assistant** that lets the users define automated tasks he wants to run on this computer These tasks should be fired either when a given event occurs (example: an expert with expertise in the area of webcams installation on Linux is now available online for real-time help) or periodically. The corresponding workflow to be triggered when the event occurs should be defined graphically by the user as well. ([WP11000] Personal workflow support)
27	Organizational process integration	Integration into **organizational processes** ([WP9000] DOM-01-09) **Relevant work packages**: WP10000

Table 4.6: Consolidated functional task management requirements: Extended functions.

5. Conceptual Task Management Model

In this section, we will describe the design of the conceptual model of the collaboration task management. The model is based on the theories described in Section 3 that provide the basis on which we build the model. Beside deeply theoretical models as for example Activity Theory we also build on more concrete task management models as described in Section 3.2. , even if these are not explicitly mentioned. Throughout the section, we will also refer to the guiding principles stated in Section 2 in order to explain specific design features.

In the first subsection, we provide the static view of the task model describing the main objects we are dealing with. Here we introduce our understanding of concepts such as activity, task, etc. and describe their principle properties. In the following subsection, we then go to the functions that are required to handle the objects described before, i.e., we provide the operational model of the task management. In the concluding subsection, we refer to the requirements as they have been identified in Section 4 and show which requirement is covered by which concept.

5.1. Basic Task Concepts

In this section, we describe the main concepts that appear in the context of the task management. They are mainly derived from the theories described in Section 3 or they result from other considerations that are then mentioned explicitly. They provide the conceptual framework for which we later described the functionality and model.

5.1.1. Personal Task Management

The Personal Task Management (PTM) is the central tool on the users' desktop that manages their tasks. The PTM includes a personal task repository (PTR) in which all tasks are stored, with which the user is concerned (see Section 6.4.1). Furthermore, the PTM provides services that go beyond the handling of a single task, e.g., to-do lists or related services.

The PTM is the central interface to the user and therefore must provide an environment that is attractive enough to the users to convince them to stay in the system and to use it.

The PTM systems of different users are related among each other in order to exchange data about work items. However, the PTM does not only provide services to delegate and organize work but also an environment that helps people to protocol their work since these work protocols provide a valuable means to help other users to accomplish their work. It is particularly the aim of this proceeding not only to gain best practice but also to show limits of existing practice. It can be assumed that it is easier to get this information during the execution of a task than afterwards.

The motivation for the user to provide this information must be that the more information the system gets the better the services provided by the system work. This especially holds for the retrieval of task related information that is based on the provided task data.

5.1.2. Activity / Action

The concepts of Activity and Action that are used in this document correspond to those developed in Activity Theory, describing an activity as the purposeful interaction of a human or non-human actor with the world in a process of mutual transformation (Leontiev, 1978). The difference between activities and actions has been described above. Since we are mainly dealing with object-orientation we will prefer the term action instead of activity. Actions can be **individual** or **collaborative**, depending on the fact whether they are executed by a single actor or group of actors. In particular, Activity Theory yields the following properties of actions:

1. They are directed towards a material or mental object (**goal**) which is specified at the beginning of the action, even it might be due to changes during the execution.
2. The action possesses a fixed **duration**, i.e., it starts and ends at specific points of time, even if there might be interrupts in between.
3. It is related to a subject (**actor**) who executes it.
4. Generally the action is mediated by tools to execute the action which might be symbolic (**information**) or material (**resources**).
5. The purposefulness of actions requires that actions are based on **plans**, where the plan describes how the goal is to be achieved by the execution.

From Psychology we know that actions can be divided into different **action phases** (Gollwitzer, 1990):

1. **Goal setting**, if a goal is not yet provided externally,
2. **Planning**, how the goal can be achieved,
3. **Enactment** of the actual execution, and finally
4. **Evaluation** of the result.

The evaluation can also be done externally if, for example, an external requester has initiated an action. The PTM must appropriately support all of these stages.

In Section 3.1.2.1 it has already been pointed out that an actor requires a specific **skill set** to be able to execute a specific type of task. Such capability related information might include

1. Specific skills,
2. Persons' role in an organisation,
3. Organisational unit for which a person is working,
4. Other information about more non-standard capabilities.

This kind of information can help to find persons that are able to work on a specific task and therefore helpful for the assignment of delegated tasks. Since this set can encompass more than mere skills we rather refer to them as **abilities**.

5.1.3. Task

The concept of task is derived from the concept of activity. However whereas the action focuses on the execution, the task describes a projected action, i.e., an action to be executed. Since we do not distinguish between actions and task with respect to the technical implementation the task includes all attributes that belong to an action, however, a task is also meaningful if not all attributes are assigned to values, e.g., for a task it is not necessary that an actor is assigned.

With respect to the actor of a task we distinguish two roles: (1) the **owner** and (2) the **executor**. The main difference between these two roles is that the owner is responsible for the execution of the task with respect to external parties while the executor is only working on the task. Both roles are explained in more detail in Section 5.1.6.

Like the action, the task essentially refers to a **goal**. How this goal is achieved, however, is open to the action, which results from the task. This means that attributes such as duration, actor, information, resources, and plan may only be partially specified. In contrast to actions, the actors of a task can change, i.e., a task can be started by one actor and can be finished by another. This means that the first actor has not completed the action while the final actor has started the action in the middle of the task. We can generally relate the goal to a concrete **objective,** which might be a document to be delivered or any other object produced or modified in the task. In contrast to the goal, which may refer to a mere activity (e.g., learn something about astronomy), the objective is a concrete object, which stands in the centre of the task activity. However, it can also refer to a specific state of an object, e.g., if a computer system shows a specific error then the objective can be to transfer the system into the state error-free. Further details to goals are also given in Section 5.1.3.3.

The **granularity** of tasks might be different. Since a task describes a unit of work that can be accomplished mainly independently (except for required input and delivered output including some exchange of messages – for details of the input and output concepts refer to Section 5.1.3.1) it does not make sense to describe all activities as separate tasks. Otherwise, the effort for users of the task management would be increased without the corresponding benefit. However, it should be kept in mind that tasks will be the units of reuse. This means that activities, which appear as part of a task, cannot not be treated as independent units of reuse but only in connection to the containing task. This independency should be the final aim of the task handling.

If we consider the task with respect to the different phases that we have found for actions, we find that a task can include **sub-tasks**, i.e., parts of the task that are related to separate tasks and might be performed by other users. The identification of such sub-tasks is part of the task planning and their initiation part of the execution. Details are provided in Sections 5.1.3.1 and 5.2.1.2, respectively.

After the execution of a task has been completed, the task becomes an information object called **case**. This notion is taken from the workflow terminology (WfMC, 1999) but the role of cases in the Nepomuk Task Management differs to some respect from the usage there. In the Nepomuk context, we only use it to distinguish between task as active units of work and cases as mere information objects that might provide guidance for other tasks.

5.1.3.1 Task Structure

A task, which is not elementary, possesses a structural component formed by a scheme of sub-tasks. The execution of these sub-tasks is part of the task execution. The concept of sub-tasks is described in the next subsection.

The handling of tasks is often related to an exchange of information and resources. Actually also information can be regarded as a kind of resource so that the distinction only refers to the fact that information is treated in a particular format by the Task Management. To this end we distinguish between **information units** (IU) and **resource units** (RU). RUs are generally external to the Task Management, e.g., rooms, formatted documents, templates, etc., while IUs are treated as internal entities. In particular, this means that they are stored and administered within the Task Management while other resources are stored externally.

A task management working in this way would not include kind of social information exchange beyond delegation of tasks and the respective exchange of IU and RU. The particular way in which the communication between requesters and executors is established, e.g., via email is to be described in another document.

In the following, the general structure of a task is to be described. This includes the main data that are assigned to a task after it has been initiated.

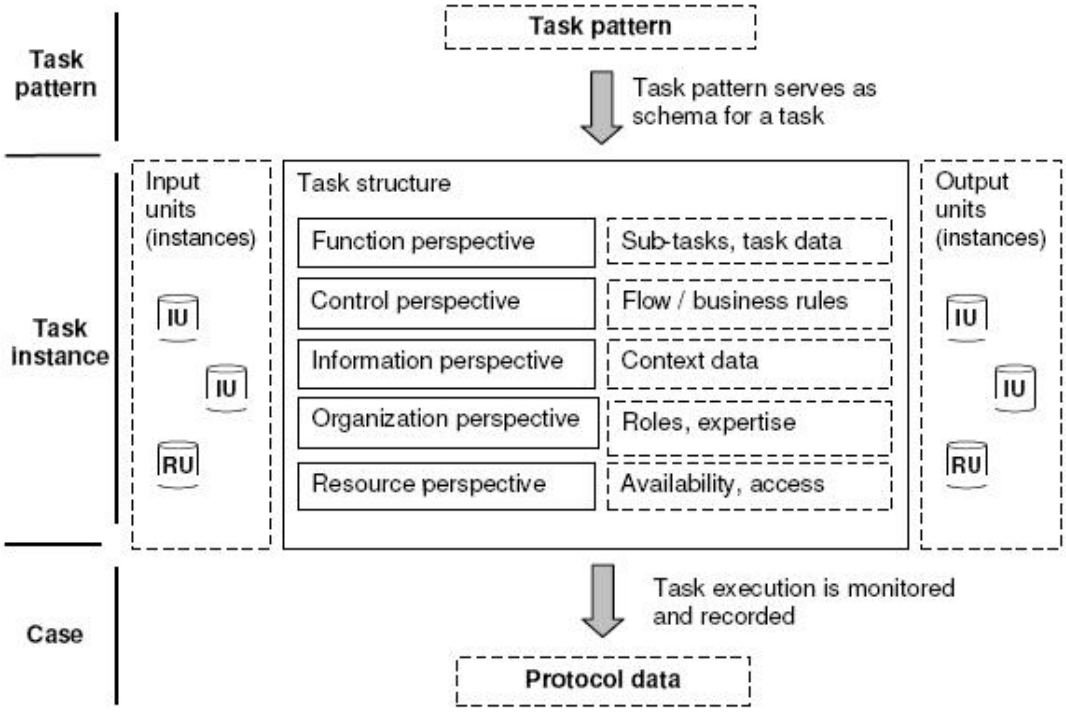

Figure 5.1: Task Structure.

The principle structure of a task after its initiation is described in Figure 5.1. The different components of the task are described as aspects, which will be explained in the following. The functional aspect describes the general sub-tasks to form the structural part of the task. During the instantiation, they will be related to task pattern that provide the template for the execution. The control aspect describes the dependencies between sub-tasks and resembles the description of

workflow models. Indeed, we can regard it as an elementary process model that cannot be further decomposed in separate tasks. The information aspect describes the general information related to the task pattern as well as the context dependent information that is added to the template. It is organized in IUs that are to be designed in a way that they contain separately usable information. For input information, this might mean that the input is supplemented by additional context information that makes it reusable. The organisational aspects mainly refer to possible and actual task owners, e.g., from former task owners in other cases, and shall support the identification of possible candidates. Finally, the resource aspect describes the RUs that are assigned to a task. During the execution of the task the data related to the different aspects are continuously updated due to the current status. After the execution has been finished, the entire information is complied in a case that is stored locally. Depending on the users modification of the pattern realized in the concrete case, these users are encouraged to update the task pattern according to their changes. Details regarding the proceeding will be provided in the following.

After the execution of a task has been completed, the task becomes an information object called **case**. This notion is taken from the workflow terminology (WfMC, 1999) but the role of cases in the Nepomuk Task Management differs to some respect from the usage there. In the Nepomuk context, we only use it to distinguish between task as active units of work and cases as mere information objects that might provide guidance for other tasks.

5.1.3.2 Collaborative Tasks

One person does not execute most tasks exclusively but by a group of persons. Here we can distinguish two different cases: (1) **Separable activities**, i.e., the work related to a task can be separated into different units that can be processed by individual users; in this case, sub-tasks are used to define these units. (2) **Collaborative activities**, i.e., it is not possible to separate the activities in a task among different users so that the users have to jointly work on the task. A typical example for such a task is a meeting, even if the different participants might attend the meeting for different reasons, e.g., some as information providers and others as information consumers; they nevertheless share the same information space. From their different interests various roles in the task can result as described later in Section 5.1.6. Since collaborative tasks play a central role in knowledge work, the Nepomuk task management must handle them. The central role plays the task owner (cf. Section 5.1.6.8) of the collaborative task who can invite other executors (cf. Section 5.1.6.9) and takes the responsibility for the execution of the task.

There is no principle difference between individual and collaborative tasks so that a transition from one type to the other is always possible and only depends on the specified co-workers, i.e., a task only becomes collaborative by the invitation of additional executors but does not possess a different structure.

5.1.3.3 Context-free versus Context-dependent Goals

The context-free (synonym context-independent) goal can be used by the PTM to detect specialisations of task patterns in addition to providing user guidance hints during task planning and execution on the intended

goal of the task. For example, should the user update the context-free goal from "capacity/effort planning" to "capacity/effort planning for EU projects", the system may recommend to the user to introduce a new task pattern as a subclass of the original task pattern. Alternatively, the PTM may suggest a completely new class of task patterns.

Whereas the task title and goal description are context-dependent and will differ from task to task e.g. "capacity/effort planning for WP3, Nepomuk project, Year 2", the generic goal is context-free and conveys the generic goal of the task, e.g., "capacity/effort planning". For ad hoc tasks (those not derived from a task pattern), the context-dependent task goal may be used as an initial approximation for the context-free goal. During task pattern creation, the system will copy the context-dependent goal and suggest to the task pattern creator to generalise the goal description by removing context-specific information. Furthermore, statistical information collected across all task patterns in a given task pattern hierarchy may provide additional insights on the needs and behaviour of users in different situations.

5.1.4. Task Relations

We can connect tasks in various ways. The most prominent task relation is the **sub-task relation**. It describes the relation between a **super-task** and a sub-task where the **sub-task** encapsulates a part of super-task that can be executed independently. This sub-task can then be delegated to other users for execution. This and other relations provide connections between different tasks that support the exchange of information and resources.

However, there can also be other relations beside the sub-task relation. For example, relations can be introduced to enable the mutual exchange of information between two tasks, which are actually independent.

5.1.4.1 Sub-task

A sub-task is an independent task, which refers to another (requesting) task via a sub-task relation, i.e., this means that its execution is required in order to fulfil the execution of the requesting task. The execution of a sub-task can thus be considered as part of the execution of the requesting task. Nevertheless the execution of the requesting task does not necessarily depend on the sub-task, e.g., in the case that the sub-task only provided helpful but not mandatory support. However, in general the task will depend on the contained sub-tasks. Therefore, the execution of a task should generally only be accomplished when all sub-tasks are executed (however, the sum of all sub-tasks does not represent the complete task since this can include additional activities). If users finish requesting tasks when some of the sub-tasks are not yet completed they get a warning message and they have to decide whether they nevertheless want to complete the task. If they do so messages are sent to the responsible persons for the still open sub-tasks which inform them about this. They are no longer obliged to the requester and can continue the task under their own supervision or close it, too.

The owner of a sub-task always has a specific role with respect to the requesting task, called **delegate.** Section 5.1.6.7 describes this role. The corresponding role of the owner of the requesting task is called **requester** and described in Section 5.1.6.3. Obviously, the requesting task becomes a super-task for the delegated task.

A sub-task has to be described formally by a title, description, dependencies, etc. and must be initiated to be executed. Initiation means that a request for executing this sub-task is sent to another user. This can be done based on a suggestion by the executor or by other services. After an addressee of the sub-task has been identified, a mail is sent to this person who has to decide whether he or she accepts the request. In case of a rejection, a new addressee must be identified whereas in case of an acceptance a task is created in the addressee's PTM. From the addressee's point of view, the original task executor appears as task requester while the addressee becomes the task executor of the newly created task. The role of the task requester includes the supervision of the executor's processing but his or her view is restricted to those aspects only that are required for the supervision. In principle, the proceeding is the same whether the executor of a sub-task is identical to the requester or not. Even if the requesters execute their sub-tasks themselves, they get new tasks in their Task Management system for this execution.

Although a sub-task depends on the super-task regarding the result and the agreed deadlines, the execution of a sub-task is mainly independent of any other task since the owner of a task can freely decide on how to do the work (**autonomy of tasks**). The requester only gets insight into the sub-task as this is necessary for the execution of his or her own task, e.g., with respect to time planning. This results in two different perspectives. On the one hand, there is the view of those people who actually work on the task. They have full access to all resources in the task on which they rely. This includes all internal working documents that are not destined for task output. This is called the **internal perspective** into the task. On the other hand, we have the perspective of an external controller who has to decide on the progress that the task makes and the output that it produces. We call this the **external perspective** onto a task. In this context, we also talk about the **common information space** of a task. It consists of all information that is necessary to perform the task and is therefore accessible by the task executors. Thus, the common information is part of the internal perspective.

5.1.4.2 Task Hierarchies and Processes

Processes appear in the Nepomuk task management only dynamically as hierarchies of tasks connected by sub-task-relations. The processes are not derived from process models except for the fact that task patterns provide local, i.e., related to an individual task, guidance which sub-tasks might be appropriate for a certain type of task. However, the respective sub-tasks are completely free in the choice of own task patterns, i.e., independently of the fact which task patterns the superior task has applied.

If we consider the structure that results by the connection of tasks via sub-task-relations we find a **task hierarchy** resulting from the task and their respective sub-tasks. They represent the division of work resulting from delegation. Nevertheless the tasks in these hierarchies can be connected to arbitrary other tasks via additional relations, which, however, do not determine the primary work process.

A certain exception of this strict hierarchical concept is the case that a task is defined as **joint sub-task** of two different tasks, i.e., it is in a sub-task relation with two requesting tasks. However, such joint sub-tasks only appear occasionally. The reason to form a joint sub-task could be that two tasks contain a common unit of work, e.g., an investigation

that is relevant for both of them. Then they can agree on a joint sub-task to perform the common work together. The responsibility towards the sub-tasks, however, remains an individual one. This means that it is possible that the result of the sub-task is accepted by the first requesting task but rejected by the second. In this case, the task would be ongoing until the last acceptance arrives.

5.1.5. Task Patterns

A central concept of the Nepomuk Task Management will be that of Task Patterns. Task Patterns describe a kind of active task templates that provide information that helps users to organize their own task. A task pattern can be regarded as an abstraction of a class of similar cases and thus describes a kind of best practice for the execution of specific tasks. In this respect, a task pattern can contain all kind of reusable information resulting from cases. In particular, we distinguish **static** and **dynamic** information provided by the task pattern. While the static information is the same for all tasks using such a pattern, the dynamic information varies with the task context. In the following, we will describe both aspects in more detail. Regarding the distinction between individual and collaborative tasks, we find no principle differences for task roles. However, the task pattern can indicate whether the corresponding task is **individual** or **collaborative**. To this end the task possesses a flag which is mainly maintained in a hidden way. By default, the task is individual but the flag is automatically changed to collaborative if further more than one executor work on it.

5.1.5.1 Static Task Pattern Information

Like a task, a task pattern is related to a specific **goal**. The goal description mainly helps users working on a specific task to identify a task pattern that can support this task. It depends on the goal with respect to which it is decided which information can be provided by a task pattern. In the following, we will describe certain types of static information that can be provided by a task pattern, even if not all of them might be implemented in the Nepomuk Task Management:

1. **Possible sub-tasks**: Due to the goal, it might be unclear whether a specific sub-task could be part of the task execution. Therefore, a list of possible sub-tasks is provided to the user from which the user can choose those that might appear as appropriate for the current task.
2. **Dependencies between sub-tasks**: The pattern can provide information about dependencies between sub-tasks. For example, it might be necessary that a user can only start with the booking of a business trip if the user has previously asked for approval for this trip. Thus, the users are pointed to activities that they otherwise might forget.
3. **Decisions**: Often similar tasks require similar decisions from which different activities result. Workflow Management is aware of this and thus decisions resulting in different activities are a central aspect of workflows.
4. **Completion measures**: To which degree a task is completed can depend on various factors. Often it depends on the type of task what is applicable. Therefore, task patterns can provide

hints which measures might be helpful for a class of similar tasks.

It is worth to mention that the task pattern only provides guidance and does not prescribe a specific proceeding. Thus, it is an offering to the user, who might take or omit this.

5.1.5.2 Dynamic Task Pattern Information

Beside the information described in Section 5.1.5.1, a task pattern can also provide information that depends on the particular context of the applying task. For example, this context is given by the goal description of the task, specific input information or context information provided by a calling task if the current task is a sub-task of this. Here the knowledge is used that tasks, which apply the same pattern, should be rather similar in their character. Therefore, we call them **similar tasks** in the following. Such information can include:

1. **Information objects**: These can be provided by similar tasks if their context has common aspects with the current task. For example, a task pattern that provides travel information can be used by tasks with different travel destinations. Tasks with more or less the same travel destination can thus provide helpful information beyond the actual pattern.
2. **Statistical information**: Due to the similarity of tasks it is also possible to derive information about the estimated execution time of a task. However, this can strongly depend on the context, e.g., a travel to a nearby location by car will probably have a completely different execution time as an air travel from one continent to another. If the particular context is known, such estimates can be determined in a much more reliable way.

To realize these services task patterns must store suitable information, e.g., such about **applying tasks**, i.e., tasks that have used this pattern. Based on this information, the task can determine all similar tasks and offer the described information.

Dynamic task pattern aspects can be described as services that a task pattern provides to support the user in planning and executing a task in a very specific way.

5.1.5.3 Task Pattern Incorporation

After an appropriate task pattern has been identified for a given task the data from the task patterns have to be transferred to the task. This process is more than just a copy since there might be some specification in the task already available so that a merge process is required. A typical example for this situation is the goal description. Since the task creation starts describing its goal, a goal description is always there. If a task pattern also provides a goal description on a more abstract level, it has to be included. Let us consider the following standard cases:

1. **Goal description** of task and task pattern will be merged into one goal description. To this end, the pattern goal description is added at the beginning of the already existing goal description of the task.
2. **Sub-tasks, dependencies, and decisions** from the task pattern will be added to the already existing data of the task.

In general, the assignment of a task pattern should always be the first step. There are two possible situations in which the user defines data that otherwise should be provided by the task pattern: (1) a suitable task pattern is not available and the user has to describe the task without any help; (2) the user has not yet looked for a task pattern but wants to keep some information for further use. In the latter case, a merge with a later added task pattern must be possible. However, this merging of tasks with respect to more complex data is difficult and Nepomuk task management will not support it. Therefore, the users should get a short warning if they create the first sub-task without using a pattern.

5.1.6. Task Roles

According to the different activities that are related to a task we have to distinguish different **task roles** where a task role characterises a specific perspective towards a task which again is related to different required functionalities and permissions. Persons involved in a task instance take one of these roles. Tasks represent encapsulated units of work and as such, they possess an **internal** view and an **external** view. The internal view is characterised by the actual work towards the task goal while the external view refers to all activities that are related to the outcome of the task but without interference. Internal und external views are consequences of the **autonomy** of tasks.

Another aspect that we address, when we distinguish between internal and external view, is that they can connected these views to specific access rights that are handled at two different levels. Referring to the external view, we regard a task as an information object among others in the PIMO. This means that we have to handle the permissions related to this view in a universal Nepomuk context. In contrast to this, the permissions related to the internal view are explicitly task model specific and therefore we must handle them within the task management system.

In the following, we will present the main task roles. They appear with respect to different groups of functionalities and permissions. Table 5.1 gives an overview of the existing task roles:

<table>
<tr><th rowspan="2">Attribute</th><th colspan="2">Internal Views</th><th colspan="2">External View</th></tr>
<tr><th></th><th>wrt. sub-tasks</th><th>involved</th><th>not involved</th></tr>
<tr><td>Controlling Roles</td><td>Task Owner</td><td rowspan="2">Requester</td><td>Controller</td><td>-</td></tr>
<tr><td>Viewing Roles</td><td>Internal Observer</td><td>External Observer</td><td>Analyst</td></tr>
<tr><td>Providing Roles</td><td>Executor</td><td>Delegate</td><td>Contributor</td><td>Creator</td></tr>
</table>

Table 5.1: Task roles.

In Table 6.2 we have categorized the considered roles according to different criteria. There are two different kinds of views, internal and external, as introduced before. Roles that belong to the former have access to the internal information space whereas those that belong to the latter only get information that is provided for controlling purposes or delivered by the task owner. With respect to the external view we can distinguish two further kinds of roles. Some external roles are to a certain degree involved in the particular activities since they depend on or are

interested in the results of the tasks. The external roles that are not involved in the task contents are those roles that are concerned with the organization of processes (analyst) or roles who are mere initiators of tasks without further interest in them (creator). Regarding the internal roles we also distinguish two types but here the criterion is whether the role originates from a delegation process encompassing sub-tasks (requester, delegate) or whether they are used in a general sense (task owner, internal observer, executor). Actually, the former roles do not describe specific right but emphasize the inclusion in a sub-task relation.

Regarding the attributed permissions, we have three degrees of access rights. The most extensive permissions are related to *controlling roles*, either internal (task owner) or external (control). However, this grouping does not imply that task owner and controller have comparable permissions. It only says that within the column the permissions of these roles are the most extended. Slightly weaker permissions belong to the *viewing role*. The owners of these roles get access to the respective information but in a more passive way, i.e., without controlling opportunities. Finally, *providing roles* only have a very limited access to the respective information space. The intention of these roles is to give access to specific information so that the respective users only get the information that is necessary to do this.

5.1.6.1 Creator (External View)

The creator describes the role of a person who creates a task, independently of the fact how this person is involved in its execution. This means that later the creators can become either task owner, if they take care of its execution themselves, or controllers, if they leave the execution to someone else but are still interested in the outcome, or none of these, if they only refer to an event from which a task results but they are no longer interested in it.

5.1.6.2 Controller (External View)

A controller is the role of a person who monitors the proceeding of a task from outside and interferes with it if this is necessary but is not involved in its execution. A typical example for a controller is a senior manager who delegates a task to another employee and only asks for the status of the task and its results.

5.1.6.3 Requester (External View)

A requester is a task creator who afterwards becomes a controller. This role usually results from a sub-task relation in which the owner of a task requests the execution of a sub-task, which is necessary for the execution of the requesting task. In this case, the owner of the requesting task becomes a requester for the sub-task.

In contrast to the creator role, however, the requester can change. For example, if a senior manager is owner for the task Budget Planning and requests the sub-task Identification of Stakeholders, he or she becomes the requester of the sub-task. However, if the senior manager delegates the task Budget Planning to his or her assistant, then the assistant becomes the requester even if the senior manager remains the creator.

5.1.6.4 Contributor (External View)

A contributor describes the role of a person who delivers resources or information to a task without being directly involved in it, i.e., the contributor is person has no access to internal information space of the task. An example of a contributor is the owner of a sub-task who takes care for the delivery of some outcome necessary for the requesting task without being involved in it.

Even if the owner of a sub-task is the most typical example of a contributor, it is possible that contributions can also result from completely different tasks, e.g., such which are only related to the current task and where the delivery is not required.

5.1.6.5 Analyst (External View)

Analyst is the role of a person who is not interested in the actual execution and the results of a task but who, for example, monitors the task to get reusable information or to analyse the cause of problems that occurred during the execution of the task. A typical example of an analyst is a process engineer who looks at a series of actual processes and their tasks to find out whether and which structures they can improve.

5.1.6.6 External Observer (External View)

The role of an observer is similar to that of an analyst; however, in this case the respective person is interested in particular results of the task. For example, the owner of a task knows that another task is working on a similar topic and is preparing a report. Just in case, that this report might be relevant for the own task the task owner wants to get informed about the report resulting form the other task. In this case, he or she becomes an observer of the task.

5.1.6.7 Delegate (External View)

The delegate describes the role of a person who receives a Task for execution.

5.1.6.8 Task Owner (Internal View)

Task Owner is the role of a person who is in charge for the successful completion and coordination of a task. The task owner has complete access to all information and resources of the task and supervises its execution. Moreover, the task owner is the contact person for persons in roles related to external views. This means that the task owner receives external messages and distributes them internally or delivers results to the outside world. Furthermore, he or she requests external information or resources that are required for the execution of the task.

The task owner can transfer the actual work on a task to an executor. In this case, the task owner remains controller for the task while the role of the executor is transferred to the addressee. If the task owner remains executor, the task becomes a collaborative task since there is more than one person working on it.

5.1.6.9 Executor (Internal View)

Executor is the role of a person who works together with the task owner on the execution of the task but has not external obligations or possibilities of interference. The executor has access to the internal workspace of the task.

5.1.6.10 Internal Observer (Internal View)

In contrast to the external observer, the internal observer can access the internal information space although he or she is not involved in the execution of the task.

5.1.7. Task States – Status Information on Tasks

The goal of the task state model (as depicted in Figure 5.2) is to distinguish phases of a task that require phase-specific activity by the different users involved in this task according to their role. This can be accompanied by provision of specific functionality by the environment or by permissions granted to the users involved. To some respect the task states correspond to the action phases introduced in Section 5.1.2, e.g., the status completed in Figure 5.2 requires the evaluation of the task related action by the controller. The aim of describing task states is to ensure a coordinated processing.

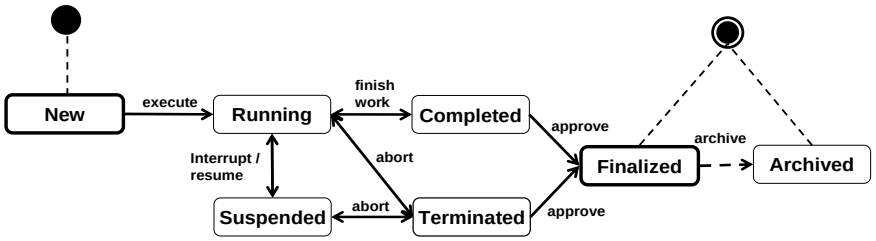

Figure 5.2: Overview of Nepomuk task states.

The Nepomuk task states are designed with the following requirements:

- Simplicity – The Nepomuk task state model is an extension of the minimalist model proposed by Grebner (2006), incorporating only one new state – Archived – borrowed from Caramba (Dustdar 2004). This reduces the user's cognitive load and enhances usability.
- Support of task roles – The model must support the task roles described in Section 5.1.6. Furthermore, the model should elucidate the relationship between task states and task roles especially in respect of state transitions.
- Execution transfer – Unlike existing models, the Nepomuk task state model should support the transfer of running tasks. A typical example is the transfer of a task to a co-worker when the task requirements are better defined e.g. the co-worker has better-suited expertise or the necessary resources (more time).

In particular, the state is visible to controllers to give them information about the progress of the task.

A task that is created starts initially in the state **new**. This means that the task formally exists and it is possible to add information to it but there is not yet a person assigned who has started to work on it.

The new task remains in the PTM of the creator until he or she initiates the task by assigning an executor. When a potential executor has accepted the task the state changes to **running**.

If the work on the task is interrupted due to external factor, e.g., if the owner is waiting for some result, and cannot work on the task the state changes to **suspended**. This means that the owner waits for a specific event that enables him or her to continue. When this event occurs, the state changes to running again and the owner receives a notification of it.

If the owner of the task decides that the goal of the task is achieved, the state changes to **completed**. Controllers are involved in this event, i.e. the controllers are notified about this event and can check the deliverables of the task. If one controller rejects the results, the state changes to running again.

Likewise, if a task is stopped before its regular end is reached, e.g., before the required output is provided, the state changes to **terminated**. Under certain conditions, the task can then be resumed and changes to the state running again.

When the last controller has accepted the results of the task, the state changes to **finalized**. This means that all involved parties agree that the work on the task is completed. Now the task can be **archived**.

Transition name	Target task state	Task function	Description of task transition
Create	New	Create task	A new task or sub-task is created with the minimal number of mandatory fields filled.
Execute	Running	Task planning Accept task Resume task	Task planning activities (including assignment of executors, acceptance of task transfer requests and resumption of suspended tasks) transition the task state into the Running state.
Interrupt	Suspended	Suspend task	Task executors can suspend the task at any time, say due to task dependencies or coordination with task contributors.
Finish	Completed	Finish task	A task is marked as completed if its goal is achieved and all required outputs have been produced.
Abort	Terminated	Terminate task	A task is terminated if it is no longer required or its goals is no longer relevant.
Approve	Finalized	Approve	A task is marked as finalized if its Controllers have accepted the task outcome or agreed that the task should be terminated.
Archive	Archived	Create or update task pattern	A task can be optionally archived by transferring selected work experience aspects to a task pattern.

Table 5.2: Nepomuk task state transitions.

Table 5.2 describes the Nepomuk task state transitions and the respective task functions.

5.2. Task Functions

In this section, we describe the operations that are related to tasks, task patterns and other entities introduced before.

5.2.1. Core Functions

The core functionality comprises all functions that are related to the basic features of a collaborative task management, i.e., a task management that essentially supports the joint and coordinated work of different users on related tasks. This includes the creation, planning and execution of tasks as well as task delegation and transfer. The following subsections describe the details.

5.2.1.1 Task Creation

Every user can create new tasks. This can be done directly from the PTM or from a suitable application that provided a plug-in for the task management. If a user creates a task then the following steps are mandatory:

1. Provide a **task title** that allows distinguishing the task from others in a task list.
2. Provide a **task goal** that is to be described by a text and might be supplemented by an assignment to a specific task category provided by a task goal hierarchy. The task goal is the minimal information required to provide some semantic information about the task. For example, the system uses the description of the task goal to **identify a task pattern** appropriately, if no other information is available. The system can support this by providing a goal hierarchy, from which the user can choose a suitable abstract goal class.
3. Provide an **owner** for the task. Every task must possess a responsible person who takes care for the processing of the task. As initial task owner, the creator of a task is assumed.

Further information might be helpful but is not required in order to keep the handling of task simple, e.g., if the user only wants to create a reminder for work to be done and plans to give further description later.

5.2.1.2 Task Planning

As we have seen before, planning is a central aspect of activities and tasks. Nepomuk tasks support planning in the following aspects:

1. Describe **sub-tasks** as steps in the current task that might be executed by other users. As an initial specification of a sub-task a goal is to be provided that later becomes part of the goal for the task that might result from this sub-task.
2. Furthermore, it is possible to describe **dependencies** between sub-tasks. These dependencies can be related to the **order** in which sub-tasks are to be executed but they can also include more general aspect, e.g., regarding a notification for a sub-task if another sub-task is completed or a transfer of its results.

3. The planning can also include the **output** that is to be produced. If a task originates from another task as sub-task, it might be that the output description is provided by the requesting task and cannot be planned freely. However, it is always possible to plan additional output.

5.2.1.3 Task Transmission

If a task is to be transferred to another user, a negotiation process has to be started in which the addressee is asked whether he or she accepts the task. The **transfer** is not completed before this addressee has sent the acceptance to the sender. This also includes the transfer of sub-tasks, called **delegation**, since these are initially assigned to the owner of the superior task. Task transfer means that the responsibility for the task at hand is completely transferred to the addressee. In the case of a sub-task transfer, only the controller role stays with the requester as the originally responsible person.

In contrast to the task transfer, the **task invitation** does not include the transfer of the responsibility but only gives other users the possibility to work on the task. In this case, the task owner stays the same but an additional user is included in the task as executor. The originally responsible person remains responsible towards controllers and other external parties. Both task transfer and task invitation we call **task transmission**.

5.2.1.4 Task Negotiation via Email

The transfer of tasks as well as the invitation to tasks requires a **negotiation** between the requesting party and the addressee. For example it must be clarified which input and output is required, whether the addressee has time and the required competency to work on the task etc. This is related to a communication process that should be supported by **email**. The requester starts the process by initiating the generation of a standardized email, which asks the addressee whether he or she is willing to accept the task as task owner or executors. In the same way, the answer to the requesting email should be more or less automatically generated. The input that is required for the addressee to decide on the acceptance can be made available to the addressee by the same email. Finally, an acceptance by the addressee must automatically trigger a transfer or generation of the task in the addressee's PTM so that no additional activity by the two parties is necessary in this respect.

5.2.1.5 Task Execution

The execution of a task cannot be clearly distinguished from the planning phase, since generally continuous updates of the plan are necessary. The execution is supported by the **task description** that provides information on how to proceed.

A central part of the execution is the **delegation** of sub-tasks. However, the execution of a task often goes beyond the execution sub-tasks since coordination efforts are generally required.

The task execution also includes **administrative activities** such as the invitation of executors or external observers. However, the PTM has to be

built in such a way that these administrative activities are reduced to a minimum.

The execution also encompasses **decision-making**. Since fundamental decisions essentially influence the proceeding of a task, it is important to protocol them in a particular way.

All activities related to the execution will be recorded. On the one hand, this is done in order to give the task owner a full overview of what happens in a task. On the other hand, the recording provides material that later helps users find information on how to proceed with their own tasks. The next section provides details.

5.2.1.6 Task History

All activities related to a task are stored in records and build up the task history. The task records are used to protocol the execution of a task beyond the completion of sub-tasks. Task records are also to be used to protocol problems that occur during the task execution. **Record types** such as 'problem' are used to give a better semantic specification of records. Recording of the task activities is a fundamental precondition for a successful later reuse of the experience gained in the course of the task. The records contain structured as well as unstructured content, e.g., descriptions of specific problems that appear during the execution of a task. The records also serve as a protocol of the task that later helps a user to provide evidence for what happened during the task.

An overview of the kind of information that is to be recorded is compiled in the following list:

1. **Changes of task states**;
2. **Task transfers**;
3. **Invitation of executors,** their **acceptance,** and their possible **withdrawal** from the task. This also includes role changes;
4. **Decisions** that have been taken in the task. Here the record serves as protocol. A decision record also includes the persons who have taken this decision and the alternatives that were available.
5. **Execution times**, which a user can specify manually or can be directly extracted from calendars. This entry might also include a description of what has been done in a short form;
6. **Problems** that occurred during the execution and the measures that have been taken to resolve them;
7. **Delegation** of sub-tasks. This also includes the recording of the completion of sub-tasks;
8. **Planning data** that have been changed such as the creation or deletion of sub-tasks and changes of dependencies. In particular this also includes the case that planning data are copied from a task pattern;
9. **Emails** that have been sent or arrived and that are related to the task;
10. **Input and output** that have been provided or delivered.

Whether there is a description of entries in a formal way or in textual form should be open for extension as well as the entry types. The given list only provided an overview of possible tapes. All entries require

additional information that is the same, independently of the particular type of entry:

1. **User** who made this entry;
2. **Timestamp** of the entry;

Most of the information should be recorded automatically without any user interaction. However, for some information, e.g., the description of problems, it seems to be preferable that the user provides the contents.

The task owner must have access to the task history and needs a particular monitor for this. The records are maintained as a list, the viewing of which can be supported by different filters. Moreover, it should be possible for a user to relate certain records to others if these are connected by their topic. This allows users to go back in their task history and find the direct line of activity from which a record results. The PTM should support **relations** between connected records, e.g., the planning of a sub-task and its execution.

5.2.1.7 Task Tracking

Users who have the role of a controller (cf. Section 5.1.6.2 need functionality to investigate the state of the task that they control. In particular, this concerns the case that they are requesters (cf. Section 5.1.6.3) and rely on the timely finalization of a sub-task. In order to observe critical situations, which might affect their own work, as soon as possible it is important to monitor continuously the proceeding of such tasks. The Nepomuk task management must provide the required functionality.

Unfortunately, it is a rather complicated problem to identify upcoming problems in an early stage. In the worst case the problem is only recognized when the result is not delivered at the projected due date. Such situations should be avoided if possible. The following indicators might give some insight on the status:

1. **Completion of sub-tasks** considering their number and different complexity
2. **Completion rate**, if the task owner can provide this. This is particular important if the task does not include any sub-tasks. However, here the problem is that it is often difficult to specify to which degree a task is already completed.
3. **Execution rate** is a quantitative measure for the completion of the task based on estimated execution time. This requires that such an estimate is available and that the task owner accurately records the time he or she worked on the task.
4. **Completed deliverables**, if several deliverables are part of the task output, also indicate the completion state of a task and should be provided if possible.

A single standard cannot describe to which degree a task is actually completed, but this strongly depends on the kind of task. Therefore, we need a variety of possible indicators.

The user has to specify which of these indicators might be applicable to a specific task but task patterns can also provide suggestions, since similar tasks usually allow for similar measures.

The monitoring surface can also include functionalities to communicate directly to the respective task owner if there are warning indicators. For

example, a message could be created that summarises the status from the controller's perspective including some remarks and asking the task owner for further information.

5.2.2. Task Pattern Handling

We consider task patterns as the central medium to distribute work experience on tasks. Therefore, one of the central questions for the Nepomuk task management concerns the creation and update of task patterns out of the daily work process. Here the personal as well as the public use of patterns should be supported and both in a rather similar way. The general advantage of the usage of task patterns instead of actual task as templates has been explained in (Riss et al., 2006).

In the following, we only describe the principle approach of task pattern handling. The particular details are open to further user studies that will be part of the project and partially carried out in work package 10000.

Besides the possibility of creating patterns from scratch, patterns will usually originate from existing task instances. Since these instances also contain information specific to the situation, an abstraction process is necessary that transforms the task contents into generally applicable task patterns. This abstraction process is also necessary to eliminate privacy problems resulting from the fact that task content contains information that is closely relating to individual users. For example, during the abstraction process it is possible to substitute a concrete person involved in a task by its role. There are different kinds of abstraction, which are listed below:

- **Remove attributes** if they are not generally relevant
- Set attribute values to **default** if required
- **Exchange concepts** that might be referring to the particular task context and should be replaced by terms that are more general. Here the system can support users. In particular, the relation between general and specific terms should be stored for other cases to support the identification of patterns.

It should also be possible to bring several task instances together and derive a common task pattern from them, e.g., a conference journey task for which a flight was booked and a conference journey task for which a train ticked was bought.

5.2.2.1 Task Pattern Creation

The creation of task patterns should emanate from existing cases. This means that an executed task is taken and a process is started at the end of which a task pattern is created. This requires an abstraction process in which context dependent and personal parts of the task description are removed or generalized. The user must be supported by the system in this rather difficult process. For example, it must be clarified in advance which task attributes are relevant for the template and which have to be adapted by the user during the pattern creation. The relevant attributes have to be displayed to the user so that he or she can check their contents and adapt them if necessary. The task management system must keep the effort for the users as minimal as possible in order to encourage them to create task patterns even for personal purpose. We

can see the creation of personal task patterns as the first step public task patterns that are generally available and used.

5.2.2.2 Task Pattern Update

The update of task patterns works according to similar principles as for the creation, i.e., the user is encouraged to provide information in an unobtrusive way supported by the Nepomuk task management. The update takes place if the users has applied a specific task pattern to his or her task and has deviated from the pattern in some way. The task management system identifies these deviations and after the completion of the task the user is asked whether these deviations are contingent, e.g., due to specific circumstances, or systematic, i.e., also relevant for other users. In the latter case, the user is asked for further specification of the introduced change in analogy to those for pattern creation. In this way, the effort for the update is reduced to a minimum and does not mean an inconvenient burden for the user.

5.2.2.3 Task Pattern Retrieval

An efficient usage of task patterns can only be established if it is possible to find the appropriate task patterns for every task at hand. If a task results from sub-task delegation, the situation is easier since there is already a right task context provided by the requesting task. If a task is created from scratch, there is usually only a task description and a formal categorisation of the task available where the formal categorisation is the best fit out of a limited numbers of possibilities.

Since it is not clear which information is generally available for the identification of a task pattern it must be ensured that the pattern identification can take place on the basis of minimal data. The minimum of information that is available for every task is the goal description and the work context of the task creator (even if the latter information can be misleading due to contingent circumstances).

Another support to be provided is a classification scheme of tasks that is provided to the user to support the identification of patterns. Based on this scheme, it is possible to provide at least a rough categorisation of tasks that helps to avoid ambiguities that might arise from goal descriptions.

5.2.3. Time Management

Besides the handling of tasks, sub-tasks, and patterns, the time management of tasks is another area that requires support. The general problem is that knowledge workers usually have more tasks than they can actually accomplish. Therefore, it is important to ensure that at least the important tasks are executed in a proper manner. An efficient prioritization of tasks is crucial in this respect. However, it is also important that the users have enough time for their tasks. This requires an accurate planning that can be supported by the system. It is central for this planning that the estimates for the execution time and the administration of the due date are as accurate as possible. If these data are available, the system can check whether the remaining time is sufficient to accomplish all tasks in time. In the end, time management and prioritization of tasks must go hand in hand. Furthermore, this

requires the support of notification mechanisms and schemes to keep the user abreast with, say, approaching deadlines as well as task status changes. Therefore, they will be treated jointly in the Nepomuk task management.

If the system recognizes a situation in which it seems to be improbable that the users can accomplish all tasks, it informs the user about this fact providing a compilation of the open task. Then the user has to decide how to deal with the situation. For example, she can delegate open tasks to other users or postpone certain tasks informing the respective requester. However, it is also possible tasks are concluded as pending, i.e., the due date is cancelled since the task is only used as a reminder.

Beside tasks with a fixed time slot there are also two other types of tasks that we consider. On the one hand, we have the tasks that require immediate processing when they arrive at the addressee. For example, this can be the case if a system breaks down and must be fixed at once. In this case, other tasks must be postponed if their schedule collides with the incoming task. On the other hand, we have tasks that cannot be planned in a concise manner. For example, I plan to read a paper. Also these tasks need some kind of time management since at a certain point of time the user has to rethink this task whether it is still relevant or whether it can be deleted or archived. The consideration of these open tasks should be included in the task management as a special management task, which might be executed regularly once a week. Perhaps there might also be other regular tasks like team meetings that are usually handled by the calendar functionality.

Regarding tasks for which time slots are reserved we have to distinguish two kinds. First, we have tasks that cannot be shifted freely, e.g., if other users are involved (collaborative tasks) or if the due date is reached. Second, we have task for which a timeslot is reserved but which can also be executed later (or earlier) without coordination efforts. The time management support can shift the latter tasks if other tasks have to be executed immediately.

5.2.4. Co-Tasks

Sub-tasks of the same super-task can be regarded as co-tasks, analogous to co-hyponyms for hyponyms of the same hypernym. Those tasks are siblings to each other. By means of ontology, learning it is possible to derive such siblings. Hereby only the name of the task is regarded. For a given task name (or a set of task), other sibling words to this task name can be retrieved by the approach described in (Brunzel et al. 2006a, Brunzel et al. 2006b, Brunzel et al. 2007). This method is worth to consider in giving help in structuring ones work into sub-tasks, especially if only a rather empty task pattern repository is available. As ontology learning is supposed to overcome the shortage of explicit semantics, this approach should help to give initial hints for explicit tasks.

5.3. Task Concept Requirements Matrix

This section shows that the core task management requirements are met by the conceptual model and functionality described in this section.

Section 4.2 details the consolidated core task management requirements collated from the case studies WP8000-WP11000. Table 5.3 maps these

core requirements to the conceptual model and functionality described in Sections 5.1 to 5.2 above.

Req-#	Description	Conceptual model section
1.	Every user can **create new tasks**	Section 5.2.1.1
2.	**Task decomposition** – Define sub-tasks **Sub-task dependencies** – Create relationships between sub-tasks within a task **Define input and output** of tasks	Section 5.1.4 & 5.2.1.2
3.	Ad-hoc task planning and **flexible changes**	Section 5.1.2, 5.1.3, 5.2.1.2
4.	Support users **in structuring their email inbox** as well as the creation of new emails.	Section 5.2.1.4
5.	The **transfer** of tasks as well as the **invitation** to tasks requires a negotiation between the requesting party and the addressee.	Section 5.1.3.2, 5.2.1.3
6.	Integrate personal as well as group **information**	Section 5.1.1, 5.1.3.1, 5.2.1.2
7.	**Task patterns** as the central medium to distribute work experience on tasks	Section 5.1.5, 5.2.2
8.	**Task pattern creation** should emanate from existing cases.	Section 5.2.2.1
9.	**Task pattern update** works according to similar principles as for the creation	Section 5.2.2.2
10.	Efficient **prioritization of tasks**	Section 5.2.3
11.	**Calendar support** for tasks	Section 5.2.3
12.	**Tracking of tasks**, experiments, projects, people, and the relationships between them	Section 5.1.4, 5.2.1.7
13.	For the execution of tasks in time it is necessary that the supervisor of a task gets enough information about the **status of delegated work**.	Section 5.1.7, 5.2.1.7
14.	For each modification on the tag level, it is needed to track the **audit trail of the changes**	Section 5.2.1.6
15.	Make task information **accessible offline** in case of no online access to the task management system.	Section 5.1.1, 5.1.3.1

Table 5.3: Relation between core requirements and conceptual model.

5.4. Security

Security and privacy issues play a significant role for the task management. Therefore, we will also include security related attributes in our task management model. In general, the Nepomuk task management will handle security issue in a consistent way with other objects of the Nepomuk system so that we must deal with security and privacy issues on a more general level, beyond the restrictions of WP3. Therefore, we only mention the issue at this point without going into further details.

6. Task Management Model

In this section, we describe the ontology that reflects the representation of the concepts described in Section 5, the **T**ask **M**odel **O**ntology (TMO).

We outline the design principles for the task management model. Then we position the ontology in the context of the Nepomuk ontology pyramid. Afterwards we describe the details of the classes and relations, which constitute the TMO.

6.1. Design Principle: Data Model vs. Ontology

Modelling a user's task results in a set of attributes and values, which comprise the individual task management model. The TMO contains the definitions of the classes and relations, which are used to build this model, that is: Any object described in a task management model must be an instance of a class from the task model ontology.

Thus the TMO defines the complete language (and thus the possible range of reality which can be described) to be used to model task-related information on the Nepomuk Social Semantic Desktop (SSD).

We have to consider that this task model should not only support existing task management approaches but more generally should support the family of task management software that can be connected to the SSD. This brings up the issue of balancing the trade-off between supporting the desired functionality and avoiding biases towards a particular case. This is a generally known issue on engineering ontologies (Spyns et al. 2002). The aim is to create a model, which can accommodate the majority of information handled by task management software.

The explicit definition of the TMO makes transparent the data structures used within the task models. Furthermore, the TMO is fully extensible. That is, every user who sees the need to expand the modelling capabilities is free to add new derived classes to the ontology on the individual desktop. While this will allow the introduction of new and additional representations, every system adhering to TMO as described here will still be able to perform the functionalities described in this document on the resulting representations.

To represent the task model ontology we employ the **N**epomuk **R**epresentation **L**anguage (NRL). NRL is based on RDFS and introduces additional elements, named graphs and view definitions being the most prominent extensions. The TMO, however, currently does not use these extensions, but only relies on the RDFS modelling and the extensions implemented in the Protégé ontology editor.

6.2. The Nepomuk TMO in the Context of Nepomuk Ontologies

In this section, we will explain the relation between the TMO and other ontologies within Nepomuk.

The TMO will not exist in isolation. It will co-exist together with other ontologies on the SSD. Since this is the case, the task ontologies reuse concepts, which are to be modelled elsewhere. Figure 6.1 shows the Semantic Desktop Ontologies Pyramid. This architecture arranges a number of ontologies wrt. their sharing scope and their degree of

stability: The representational layer provides concepts which are to be used by anybody who models an entity in a NEPOMUK semantic desktop – those concepts (the NRL in particular) must thus exhibit both a large sharing scope and maximum stability. At the other end of the pyramid, Nepomuk forsees the generation and management of the so called **P**ersonal **I**nformation **Mo**del **O**ntology (PIMO), cf. Sauermann (2006). Here the individual user is free to represent whatever concept seems useful in the context of a personal desktop; strict formality or sharing with others are not necessarily requested here. A PIMO rather reflects the personal view that a SSD user has upon the world, i.e. the domain. Consequently, such concepts will exhibit low sharing scope and low stability, as the user may modify at whim.
TMO

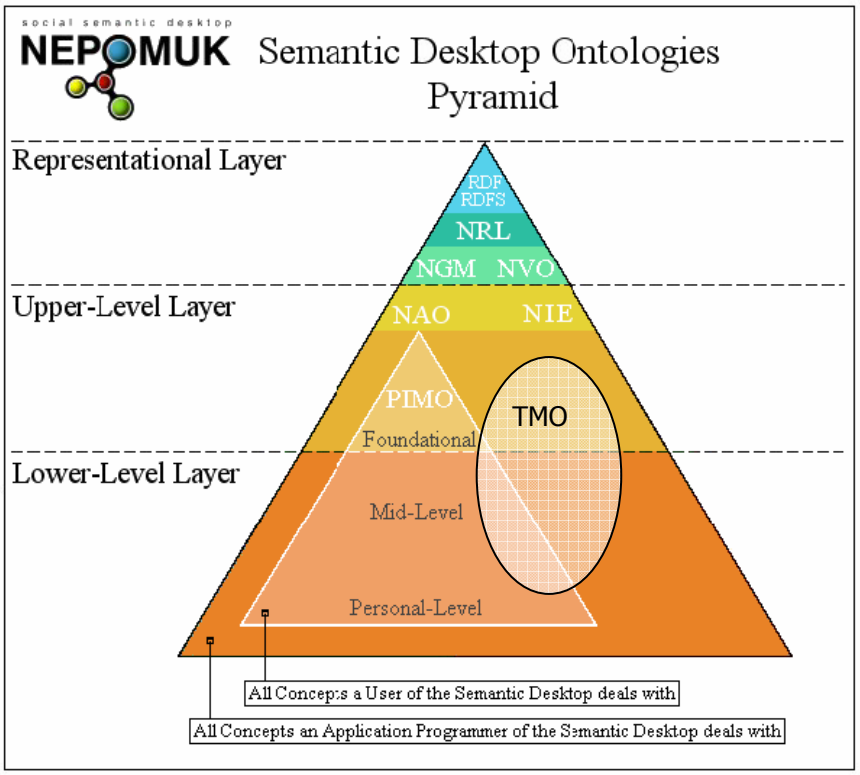

Figure 6.1: Semantic Desktop Ontology Pyramid (NRL 2006).

From the mentioned ontologies in Figure 6.1, PIMO and NRL are relevant from a task management point-of-view. As already stated, the TMO like all other Nepomuk ontologies will be represented in NRL.

The TMO is a natural part of the PIMO, as the user is free to represent, document and annotate task-like information in any way which seems suitable. However, some of our conceptualizations go beyond the personal interest. In particular, all concepts related to communication and exchange result in broad sharing and require higher stability. Application programmers may build on such concepts in order to realize the exchange-oriented functionalities. In summary, TMO is part of the domain models and overlaps with the PIMO.

The specification of the TMO makes use of a number of concepts which are supposed to reside within the PIMO, e.g. Person. For the explanation of the TMO, we take their existence within the PIMO for granted. The details of how such a concept is modelled are explained elsewhere.

6.3. The Nepomuk Task Model Ontology

In the following, we will describe the TMO. We will provide a textual description of the most relevant aspects of the TMO. The formal modelling was performed with Protégé 3.2 in such a way to be compatible with the Nepomuk NRL (NRL 2006 and Sintek et al. 2006) semantic model.

This section includes several diagrams where the conceptualization of the task model is visualised. The diagrams focus on certain aspects of the task model. They enable the reader to view a part of the conceptualization directly in this document, without the need for switching to Protégé editor to see the concrete model. The Protégé plug-in Ontoviz (Sintek 2001) generates the diagrams. Not every piece of information visible in the diagrams is explained in detail in the text as the diagrams have a documentary character.

Section 6.3.1 will be devoted to the task concept itself. The subsequent sections will describe the model for task transmission. This is followed by operations, which should be performed with and upon tasks.

Note on convention

In the following sections, attributes are presented using the following schema: `[attributeName: attributeType] minCardinality:maxCardinality`. For example, the notation `[name:String]1:1` represents the attribute "name" having the type `String` and it has mandatory to occur exactly one time. A `*` denotes an arbitrary number of occurrences.

6.3.1. Task Model

This section is about how the task model was "implemented" as an ontology. We will show classes as well as the attributes, which interlink those classes. Section 6.3.1.1 will show the core attribute set which is regarded as "core" to the task model. Then we show attributes and classes, which are important for different topics.

The following table lists all attributes of the class `NepomukTask` and the section where the attribute is explained.

Attribute Name	Reason for Existence of Attribute (reference to sections if possible)	Section where Attribute is described
creationDateTime	5.2.3 Time Management	6.3.1.1 – Core Attributes
hasAbilityCarrierInvolvements	5.1.2. Activity / Action	6.3.1.3 - Task execution: Ability Carriers
hasAttachments	5.1.3.1 Task Structure - information units (IU) and resource units (RU)	6.3.1.4 - Attachments
hasContextDependentGoal	5.1.3.3 Context-free versus Context-dependent Goals	6.3.1.1 – Core Attributes
hasContextIndependentGoal	5.1.3.3 Context-free versus Context-dependent Goals	6.3.1.1 – Core Attributes
hasInvolvedPersons	5.1.6 Task Roles	6.3.1.2 - Person Involvements
hasLogEntries	5.2.1.6 Task History	6.3.1.7 - History
hasNotification	5.2.3 Time Management	6.3.1.5.3 – Notification by Reminders
hasTaskOrderingParadigms	5.2.3 Time Management	6.3.1.5.2 – Task Ordering Paradigms
hasTaskSources	5.2.2 Task Pattern Handling	6.3.1.6 – Task Source
hasPlanningAndTrackingInformation	5.2.3 - Time Management	6.3.1.5.1 - Planning and Tracking of Time and Progress
hasTypeOrCategory	5.2.1.1 - Tasks are categorized in order to identify task patterns (Section 5: formal/informal)	6.3.1.1 – Core Attributes
id	5.1.3. Task – Not explicitly mentioned there, but there the task (e.g. the goal) as such is introduced	6.3.1.1 – Core Attributes
name	5.1.3. Task – Not explicitly mentioned there, but there the task (e.g. the goal) as such is introduced	6.3.1.1 – Core Attributes
privacy	5.4. Security	6.3.1.1 – Core Attributes
state	5.1.7 Task States and 5.2.1.7 Task Tracking	6.3.1.1 – Core Attributes
subTasks	5.1.4 Task Relations and 5.1.3.1 Task Structure	6.3.1.8 Sub-tasks
superTasks	5.1.4 Task Relations and 5.1.3.1 Task Structure	6.3.1.8 Sub-tasks
taskDescription	5.1.3. Task – Not explicitly mentioned there, but there the task (e.g. the goal) as such is introduced	6.3.1.1 – Core Attributes

Table 6.1: Attributes of the Class NepomukTask (alphabetical order) and the section where the attributes are described

Additional aspects of the TMO are task dependencies, described in Section 6.3.1.9 (motivated by Section 5.1.4.2), the transmission of task and access rights described in Section 6.3.2 (motivated by Section 5.2.1.3) and task patterns described in Section 6.4 (motivated by Section 5.2.2).

6.3.1.1 Core Attributes

Core attributes represent the basic attributes of a task. There are only few mandatory attributes, i.e. the attributes that are required for a task at creation time. These are the automatically generated Task Identifier, Creation Time and the Task Name. All other attributes are optional. Table 6.1 shows and describes the core attributes.

Attribute	Description
Task Identifier `[id: String]1:1`	The Task Identifier allows a unique identification of a task object within the range of all Nepomuk objects. The Task Identifier is automatically generated during the creation of a task. The generation of identifiers (IDs) is a Nepomuk architecture issue (WP2000/WP6000).
Creation Time `[creationDateTime: DateTime]1:1`	The time a task instance was initially created.
Task Name `[name: String]1:1`	The Task Name helps the user to identify a task in a list. It should be expressive enough to enable a meaningful recognition. Details should be written in the description attribute instead. A name attribute is not allowed to contain line breaks.
Task Description `[taskDescription: Description]0:1`	The task description helps users to understand the goal and the proceeding of a task. It can also describe the context of a task. The task description is composed at minimum of a summary of what is done to reach the goal. The task description is the main source for identifying related information, e.g., suitable patterns. A Task Description can be either an informal, described textual content (→`TextualDescription`) or it can be a more formally structured representation (→`FormalDescription`). Technology considerations: Informal descriptions allow for text similarity processing, a formal description allows for applying case based similarity measures.
State `[state:Symbol]1:1`	The task state describes the current state of the task as described in Section 5.1.7.

Goals
`[hasContextIndependentGoal: Description]0:1`
`[hasContextDependentGoal: Description]0:1`
	It is possible to attach two different types of goals: context dependent and context independent goals.
	`ContextIndependentGoal` – Context-free goal of a task – emanates from the task pattern and is stored locally at the task. The task owner or executor does typically not change this unless new types of related problems or goals are identified.
	`ContextDependentGoal` – Gives more (context-aware) details in comparison to `ContextIndependentGoal`, i.e. more information that is related to the particular task, e.g. the travel location in the example of the travel-booking task.
	See Section 5.1.3.3 for further details.

Categories `[hasTypeOrCategory: Activity]0:*`	The categories depict a selection from a set of existing "Activities" of the PIMO. Category herein means the "type of the task, e.g. "Meeting" or "Conference Journey".
Privacy Status `[privacy: Symbol]1:1`	Privacy Status serves for the separation between a professional and a private purpose of a task. This attribute provides with the values "professional/private" a high-level separation of privacy in terms of setting distribution and access rights to other users for the task. This separation may arise as a general Nepomuk issue and may therefore be handled in conjunction with a privacy preserving SSD architecture.

Table 6.1: Core Task Attributes.

6.3.1.2 Person Involvements

The class `Person_Involvement` realizes the involvement of persons with detailed roles on a NepomukTask and is motivated by Section 5.1.6. Instances of the class `Person_Involvement` are attached to the `NepomukTask` by the attribute named `hasInvolvedPersons`. `Person_Involvement` consists of two attributes, a role depicted by an instance of `Person_Task_Role` and an instance of the class `Person`. This conceptualization is visualized in Figure 6.2.

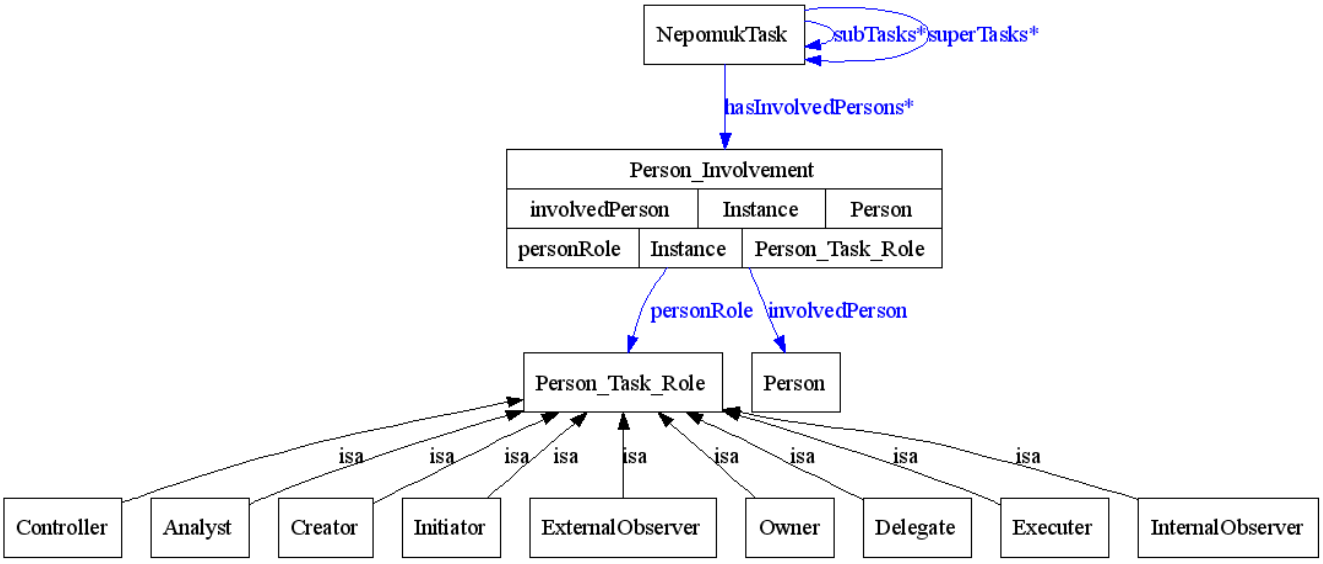

Figure 6.2: Attachment of Persons, Roles are explicitly stated.

6.3.1.3 Task Execution: Ability Carriers

The execution of a task relies on certain abilities. The abstract concept of `Abilitiy_Carriers` circumference all those more concrete concepts of which one can think of while working on tasks. Using this abstract class enables to substitute such Ability Carrier's in the process of generating patterns from task instances and vice versa in the process of instantiating task instances from patterns without violating the schema.

With this attribute, a series of ability carrying entities (`Person`, `Role`, `Skill`, `OrganizationalUnit`, `InformalDescribedAbility`) and the role of involvement (required, request, used) is enabled. The role hereby allows specifying how the ability carrying entity is or was involved.

Ability carrying entities are already listed in Section 5.1.2.

Class Name (subclass of AbilityCarrier)	Description (taken from Section 5.1.2)
`Skill`	Specific skills
`Person`	A person as a whole
`Role`	Persons role in an organisation
`OrganizationalUnit`	Organisational unit for which a person is working
`TextualDescribedAbility`	Other information about more non-standard capabilities

Table 6.2: Description of Classes that refer to ability carrying entities.

This modelling allows for a balance between being too vague and forcing to many generalizations/specializations.

The "type of involvement" further specifies the kind of involvement. It allows expressing if something is merely requested or if it is regarded as required. It also allows to manifest the "things" which have been used in conducting the task and therefore allows for development in time since what was initially considered and what is finally done may differ.

In Figure 6.3 the conceptual model for involved abilities and their corresponding roles is shown. The class `AbilityCarrier_Involvement` ties together an `AbilityCarrier` with an `AbilityCarrier_Role`.

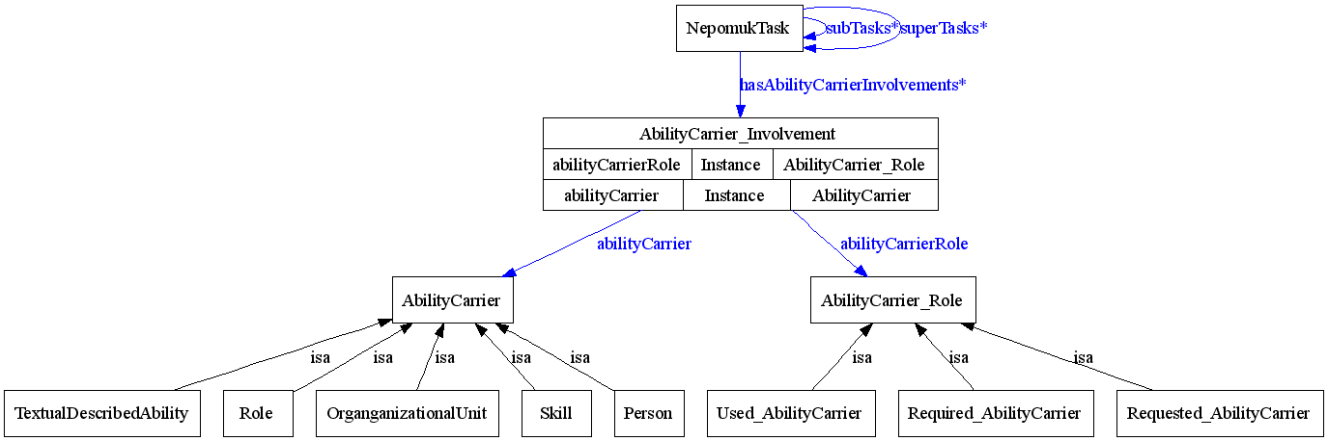

Figure 6.3: Involvement of Abilities, Abilities are further specified by a role.

6.3.1.4 Attachments

By means of attachments, references to other resources can be established. Resources are information objects. Every piece of information, which can be referenced, on the SSD is an information object. In contrast to the usual SSD references/associations, here additionally information can be specified. Further metadata about the role an attachment plays can be stated. It can be expressed what the Role of attachment is, regarding *"desired/requested"* or *"required"* or *"potentially useful / somehow related" or "used/produced/achieved"*. In addition, it can be made explicit whether something is "input" or "output". By means of those attributes, the user can separate the attachments according to his belief. Attachments are described in Section 5.1.3.1.

Class Name (subclass of Attachment_Role)	Description
`Required`	All attachments which are supposed to be necessary for the execution of the task.
`Desired_Requested`	All attachments which are supposed to be "nice to have" for the execution of the task, but which are not necessary.
`Used_Produced_Achieved`	This is to keep a record if a attachment was really used during the execution of the task – in contrast to something which was thought to be useful but which was never used. The assignment of these role is performed during and/or afterwards the task execution.
`Related`	Something where at the moment of the statement could not be assigned to the other three roles

Table 6.3: Roles, which can be assigned to, attached resources.

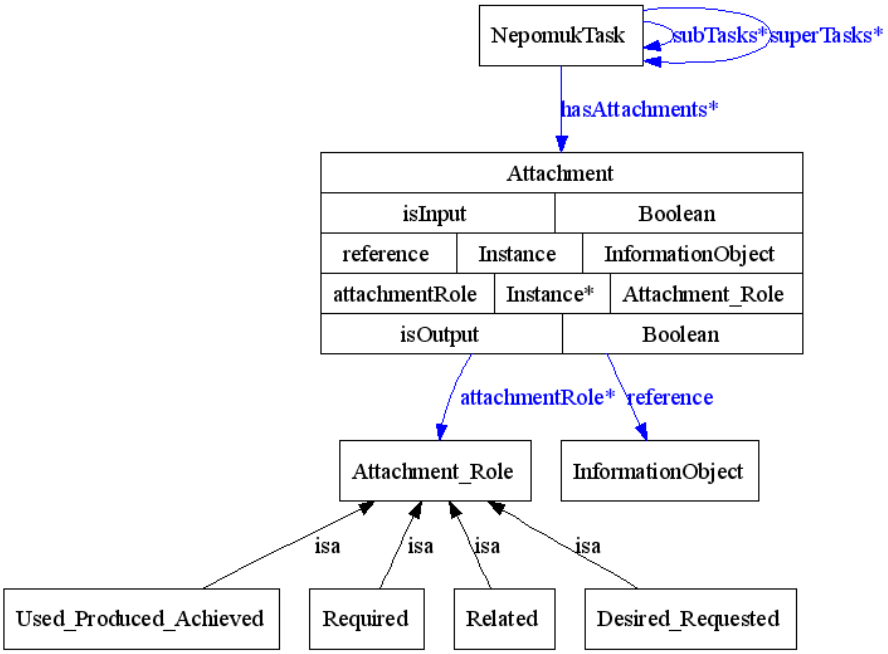

Figure 6.4: Task Attachments, specified by a role.

6.3.1.5 Time Management

The following three subsections are devoted to issues related to time management. This are the planning and tracking of time and progress, the attachment of values to order task (e.g. priority), and facilities to inform the user when certain dates approach.

6.3.1.5.1 Planning and Tracking of Time and Progress

In order to plan and track time and progress, planning, execution and finally completion of tasks can be differentiated. Those dichotomy of target/actual combined with start/end; and target/actual combined with progress (completion) and time usage are foreseen in our task model. A concrete example of what a user would like to see is depicted in the following example:

	target	actual
start	05.02.2007	03.02.2007
end	15.02.2007	-
completion	100%	65%
time used	100 h	50 h

Table 6.4: Example for the target/actual dichotomy, a user would see something like this information.

The attribute `dueDate` can accommodate a external given deadline (in contrast, `target_end` reflects the users belief when he wants to have the task finished).

The "target" column represents the information available at planning time. The "actual" column represents the more recent information, which is updated during task execution. That information together can be used for progress tracking and for consistence checking (conflict recognition / conflict de-escalation) and for a posterior statistical analysis (benchmarking/auditing). Last but no least this information can be used for scheduling a task.

Further, it is possible to keep a history if time spans when this task was worked on. This is done by a series of `TimeUsage` Objects. Additionally the person who has last updated the task and the time of the update are stored.

Figure 6.5 shows the conceptualization of planning and tracking information. As one can see, the attributes of the classes `StartEnd`, `Progress` and `TimeUsage` occur pair wise as "target" and "actual" variant.

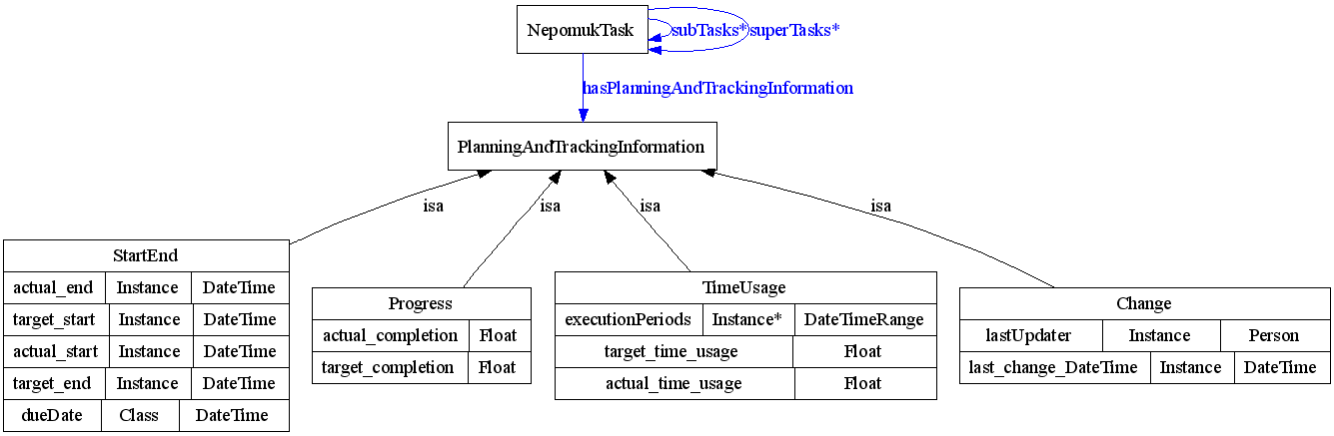

Figure 6.5: Planning and Tracking Information, most attributes occur twice, in a target and in a actual version.

6.3.1.5.2 Task Ordering Paradigms

Time management also deals with the problem of which tasks should be done next (in a narrow time window) and which tasks to perform at all. There are many tasks, which would require a work volume going beyond what can be accomplished. The selection of the subset of most relevant tasks according to manually judgments of priority/importance/urgency is one way to proceed with this circumstance. The selection of tasks, which could be delegated to other co-workers, is another possibility criterion.

Since there are different paradigms for categorizing tasks, e.g. just priority or importance and urgency, those values are encapsulated into separate classes. We do not want to force the user to commit to a specific paradigm.

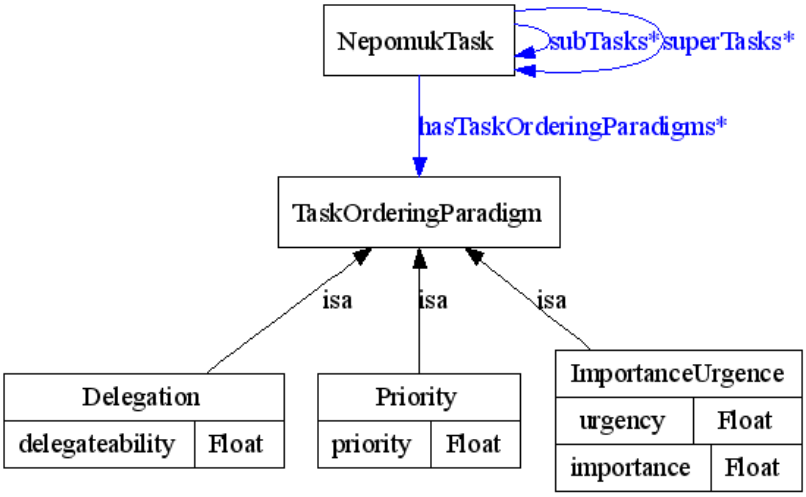

Figure 6.6: Task Attributes for stating the personal judgement of priority and other information, which is regarded as helpful in dealing with task amounts.

6.3.1.5.3 Notification by Reminders

Under certain circumstances, it is necessary to inform the user of emerging or anticipated situations (see Section 5.2.3). By means of reminders, we foresee that users are alerted to such situations. In general, notifications should be employed when necessary and kept to a minimum.

Examples of these situations include:

- There is insufficient time to complete tasks based on estimations for execution times.
- Conflict of schedules due to assignment to new tasks.
- Occurrence of ad hoc high priority or critical events e.g. system break down resulting in halt to production.

Other examples, which may be considered, include:

- The availability of new Ability Carriers for a task, which has yet to be assigned.
- The availability additional time resources as a result of task delegation, transfer or termination.

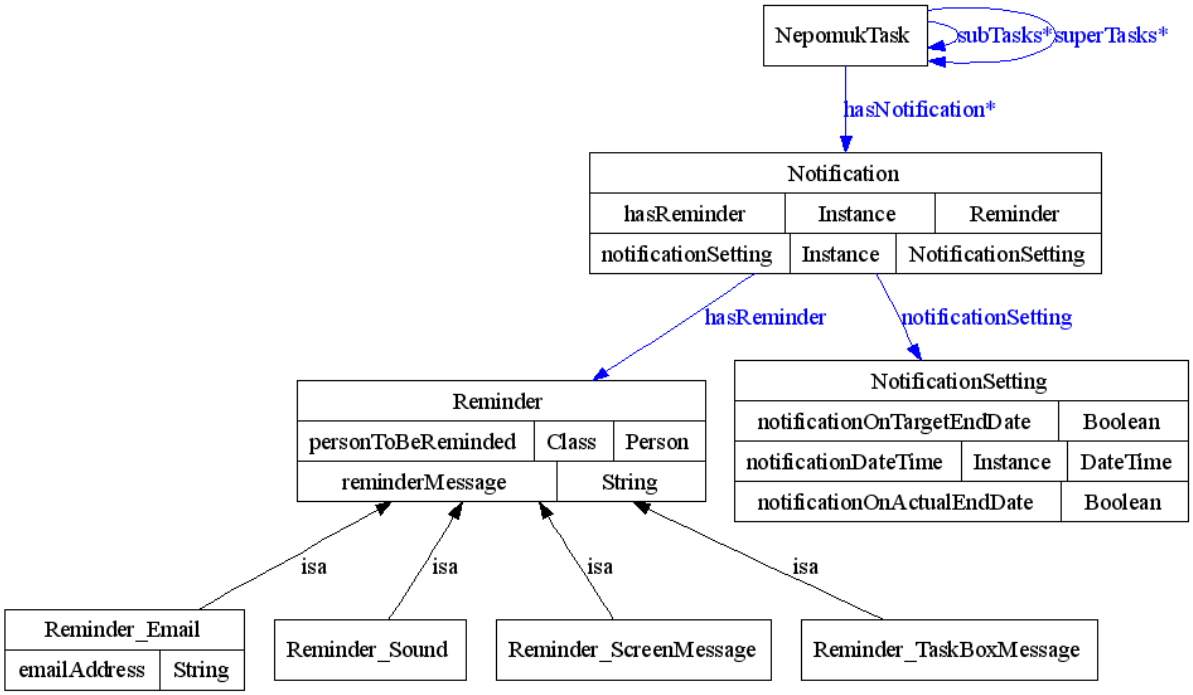

Figure 6.7: The conceptual schema by which Notification via different types of Reminders is realized.

6.3.1.6 Task Source

An important goal of using tasks on the SSD is the reuse of former task knowledge. When a task is derived from existing structures (instances or patterns) this information is kept for record. Keeping track of this information allows for further statistics on the reuse of task knowledge. Such statistics can help in maintaining the task repository and for searching patterns (see Section 5.2.2 for details).

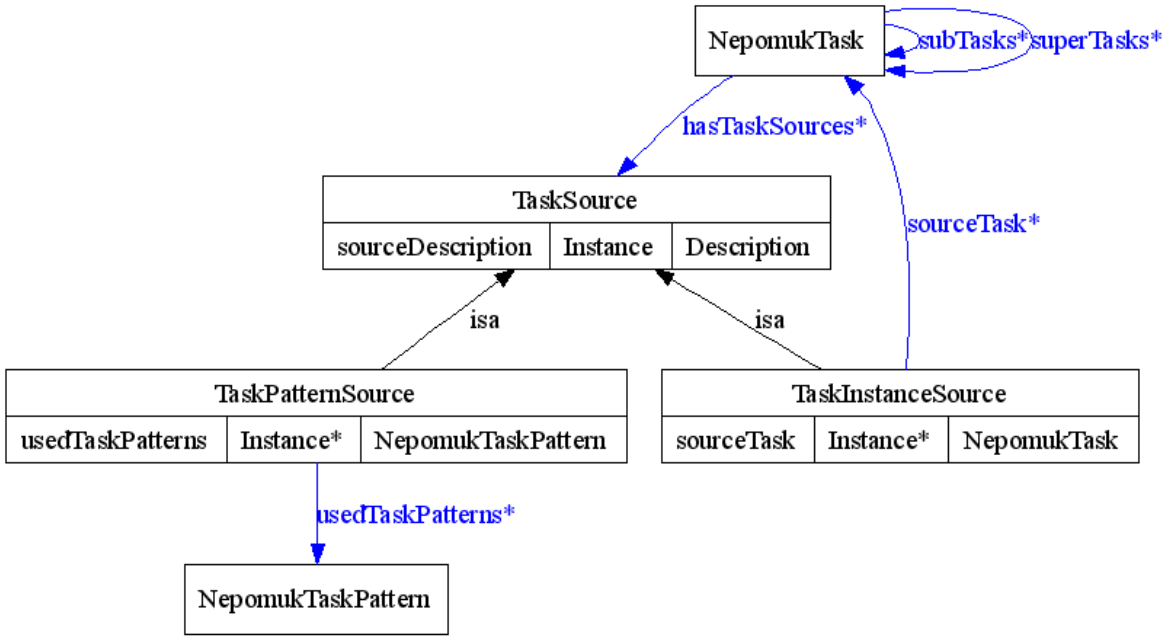

Figure 6.8: The Source of a task is kept, Task Sources are Task Instances and Task Patterns.

6.3.1.7 Task History

Since a task will evolve during its lifecycle, it is desirable to make changes traceable. To do so, the property `hasLogEntries` will aggregate a series of LogEntries (Class `LogEntry`). The abstract class `LogEntry` will have concrete subclasses where all relevant logging details are accommodated. Such relevant things, which are likely to be logged, are already listed in Section 5.2.1.6. A prominent example is to keep track of interactions by the user, e.g. like the "command history" of Protégé.

6.3.1.8 Sub-tasks

For the task management model, we want to allow for hierarchical task structures. We allow for the decomposition of tasks into sub-tasks. Tasks can be arranged in a hierarchy or the other way around: tasks form a hierarchy. In the following, we call the task which is "below" a task in the hierarchy "sub-task". The task, which is "above" a sub-task, we refer to as super-task. The super-task is the parent of the sub-task. The decomposition of tasks into sub-tasks allows to separate work items, which are rather distinguishable. The granularity of such sub-tasks may be very different, ranging from long-term task to very fine grained task in the scale of minutes/seconds work to perform.

Attribute	Description
Sub-tasks `[hasSubTasks: NepomukTask]0:*`	A task can have no or an arbitrary number of sub-tasks.
Super-tasks `[hasSuperTasks: NepomukTask]0:*`	Still subject to discussion is whether we will allow a task to have more than one super-task. On the one hand, this introduces the complexity of multi inheritance. However, there is no direct "inheritance" in the task instance hierarchy. On the other hand, there is a need for multiple super-tasks when a task is shared between several people, sincea shared task may be placed in different locations. A super-task can one side originate from the typical case, that one has created a sub-task, on the other side, a task may get another super-task if the receiver of a task accepts a task and places the task somewhere in his task hierarchy. By placing the task in the hierarchy of the receiver, it might be assigned to a different super-task than on the original hierarchy of the sender. For the time being we favour to allow more than one super-task, though this might be revised after implementing and using the task management system.

Table 6.5: Attributes stating the position of a task in a hierarchy: Super-task and Sub-tasks.

Conceptually, tasks can be expressed as shown in Figure 6.9.

Figure 6.9: The Task Schema allows for references to Super-tasks and Sub-tasks.

On the instance level, a task hierarchy looks like in Figure 6.10.

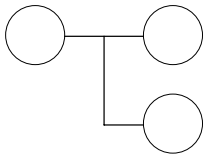

Figure 6.10: Task Instances can form a hierarchy.

Sub-tasks can also originate from reusing task knowledge from a task repository (see Section 5.2.2.3).

If no sub-tasks are used this corresponds to the usage of a flat list of To-Do items.

Additionally `TaskDependencies` described in the next section may add Metadata on the sub-tasks relation. This allows specifying additional metadata on this super-task/sub-task relation, as depicted in Figure 6.11. Those additional metadata may e.g. contain descriptions of why this sub-task was created, or it may refer to choosing the appropriate alternatives (when to choose "booking a train" compared to "booking a flight").

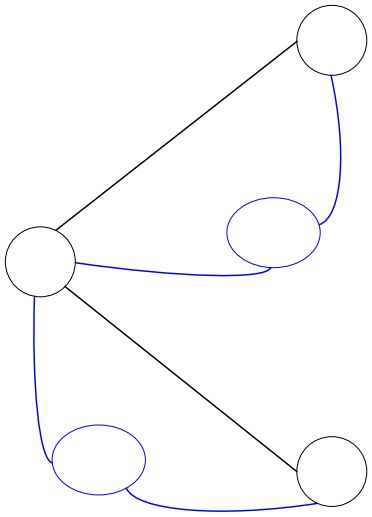

Figure 6.11: Relations between arbitrary Tasks can be added. Those relations carry metadata about the relation.

6.3.1.9 Task Dependencies

Up to now, the described task model only allows for a hierarchical decomposition. Between the tasks, further dependencies may exist. These dependencies allow for a graph network structure. For ease of use, dependencies should not be too frequent, otherwise the primarily character of a hierarchy would be diminished and a consequent graph representation would become considerable. However, such a graph representation has other drawbacks, the user is likely to loose oversight, tree structures are more helpful in structuring the work.

A dependency relation is characterized by the type of the relation and by an additional description. There are different possibilities for dependency relations between tasks.

There are directed relations, which give an ordering as

- A is precursor to B
- A is successor to B

They are realized by using the relation type of the corresponding type `Precursor` and `Successor`.

For the case of a super-task and a direct sub-task, the special type of `SuperTaskSubTask` can be set. Since "A is super-task to B " was already specified without `TaskDependencies`, this allows to give further explanations on the origin of this decomposition by means of the description attribute.

There are also relation types, which give no ordering. Those undirected relations (associations) are e.g.

- A is similar to B
- A is interdependent to B

They are realized by choosing the corresponding type `Interdependence` and `Similarity` as the relation type.

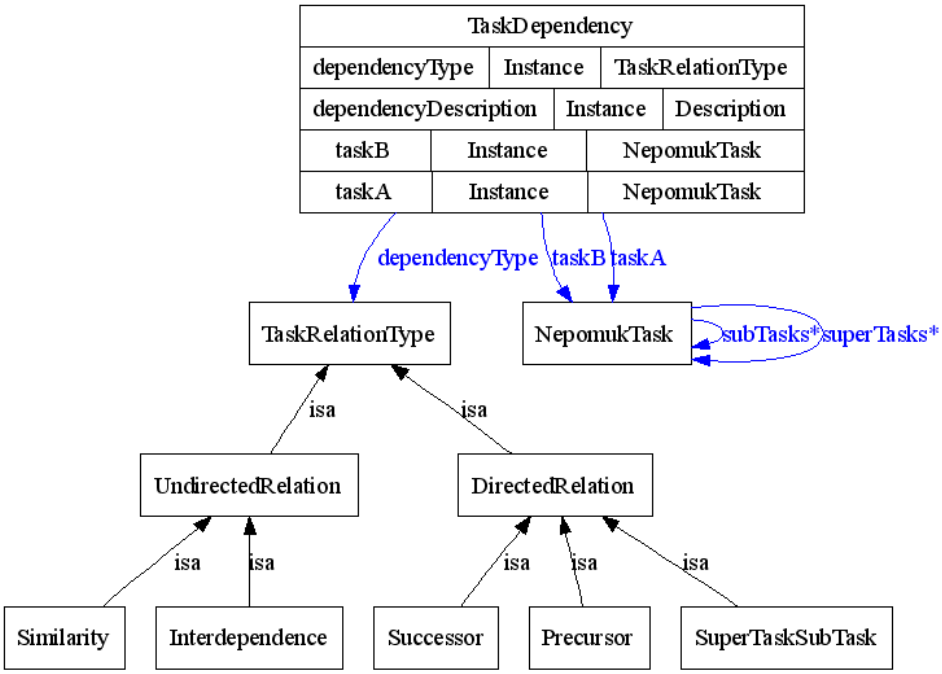

Figure 6.12: Expression of, Arbitrary Tasks can be associated by directed and undirected Relations.

6.3.2. Task Transmission and Access Rights

6.3.2.1 Task Transmission

On the SSD, tasks are not restricted to one person and may cross from the PTM of one person to the PTM of another. With transmission, we refer to the process of sending a task – from one person (sender) to one or more other persons (receiver(s)) (see Section 5.2.1.3 Task Transmission). Task delegation and task transfer are two special kinds of task transmission which are described at the end of this section. In addition, the collaborative task is realized by task transmission.

For the process of sending a task, some information is required. This information is also modelled in the task ontology. This information is still useful after the process of sending a task was completed.

For a task transmission, some metadata is generated and stored.

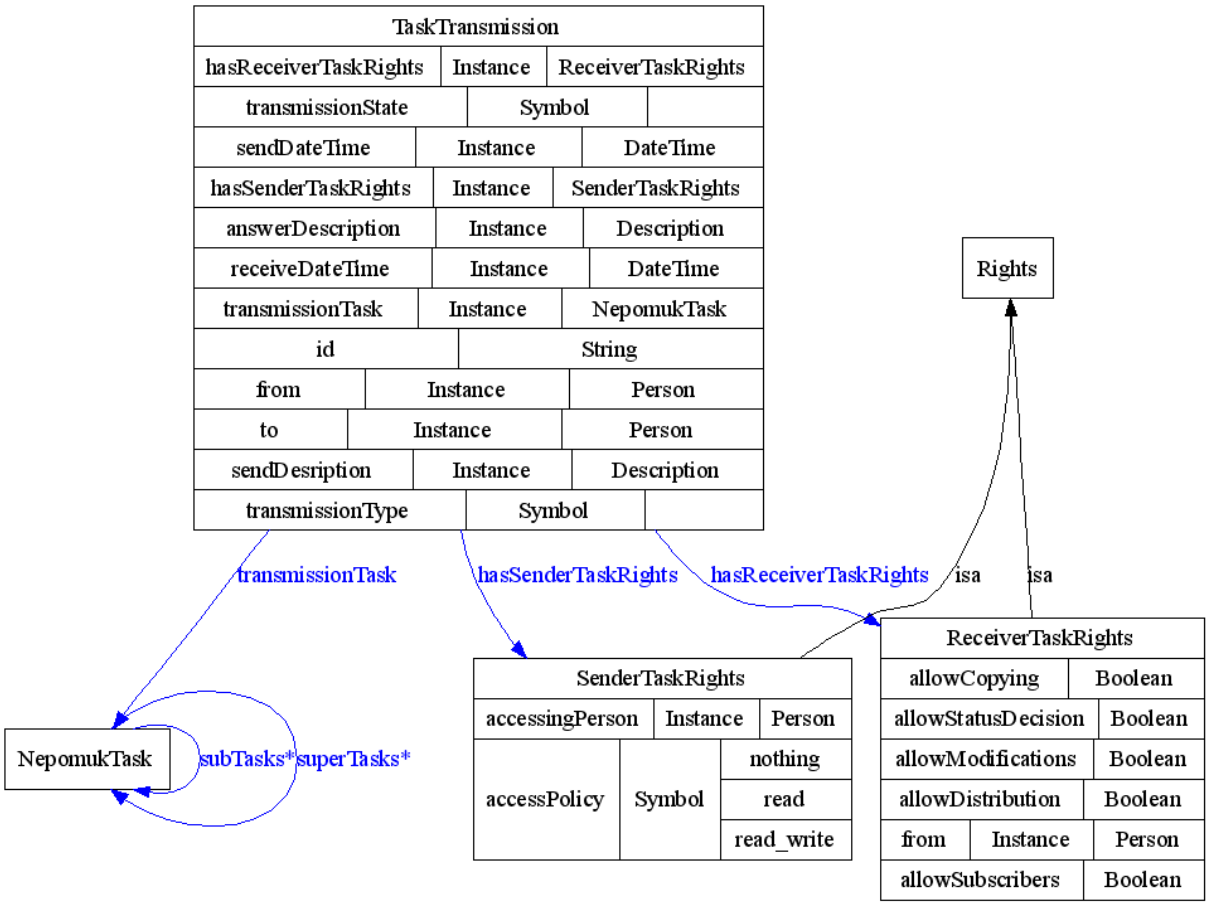

Figure 6.13: To conduct a task transmission, basic information such as "from" and "to" as well as detailed access rights are incorporated.

Task Delegation is a process where the sender of the task restricts the access rights of the receiver. This includes the right to distribute further this task and additionally the obligation to give feedback to the sender. The person that receives a task by delegation usually has not the full control about the task. The attributes described in the following section have the purpose to enable such "access rights". The receiver will also probably have obligations regarding what to report to whom at which time.

In contrast, the simplest case is that all rights are granted to the receiver and there is no feedback desired by the sender. What to do with the task may be apparent by the organization context, or it may be left to the receiver. This is like sending an email – but with the advantage that the information is transferred in the "task space" of the participating persons.

6.3.2.2 Access Rights

There are two kinds of access rights. On one side, there are the **rights** which regulate what the sender gets from the receiver, and on the other side, there are the **obligations**, which the sender puts upon the task on the receiver. Both types are used as a form of access control list, where for an arbitrary request between desktop A and desktop B can be determined what concepts are to be exchanged/synchronized or made visible.

6.3.2.2.1 Rights of the Sender on the side of the Receiver

When a task is transferred, the sender potentially has the desire to be able to track the progress of the task by having read access or the sender may also want to be able to further edit the transferred task by having write access. This access is requested (or enforced) before the task is sent (default is that there is read access but no write access).

The `accessPolicy` attribute can have one of the three following states:

State	Description
`nothing`	No access granted
`read`	Reading up-to-date information
`read_write`	additionally to read, also editing is allowed

Table 6.6: Symbolic values which can be assigned to `accessPolicy`.

With this attribute, a kind of access control list is created. This access rights are also used when additional subscribers (`Person_Involvement` with a Role `Subscriber`) are added to a task.

6.3.2.2.2 Rights of the Receiver stated by the Sender

Those attributes represent the preference on what the receiver of a task is allowed to do with the task. Depending on the context, e.g. on a strict organizational setting, the preference might be enforced as a restriction.

Right	Description
`allowDistribution`	This attribute regulates whether the receiver is allowed to send this task to other persons.
`allowSubscribers`	Determines if the receivers is allowed to make this task visible to other persons.
`allowModifications`	This regulates if the receiver can make changes to the attributes of the task. E.g. This does not prohibit all modifications, since the SSD aims for enabling the user to use his information.
`allowStateDecision`	In certain settings, only the task owner may be allowed to undertake a decision on the state of a task. With this attribute this functionality is enabled.
`allowCopying`	This is important if the receiver a task wants to reuse this task - with or without a pattern repository. If copying is allowed, reuse by (re)instantiation is enabled.

Table 6.7: Boolean Attributes which are given by the sender of a task to the receiver of a task.

6.3.3. Task operations

Task operations represent a system-specific perspective on task functions (see Section 5.2.) that work on the task model.

Keep in mind, the goal of this deliverable is the creation of the Nepomuk Task Management Model. Despite the task operations are not core of this model, the described functionality helps to understand the Nepomuk Task Management Model.

Table 6.8 lists the operations which may be conducted within the PTM. However, these operations are not conclusive and may change during the

analysis and design phases. We do not show those functions which do not directly originate from the task model as e.g. addPerson(Role, Person).

Functionality Field of Operation	Pseudo Operation Name	Description of Operation
Task Transmission	show Task Transmission Dialog	A dialog should be shown to the user where he can enter the information related to a task transmission. The user might finally decide to send the task or abort the transmission.
	perform Task Transmission	By means of SSD communication facilities, the task transmission is performed.
	show Dialog For Received Task	If a task is received from somewhere, a dialog where the user can accept or reject a task is shown. This dialog might only be open if the user explicitly looks in his "task inbox, so not to disturb the users frequently.
	send Task Transmission Answer	The receiver decision is communicated back to the sender.
Search for Tasks/Cases/Patterns	perform Unstructured Task Search	Retrieval of Tasks/Cases/Patterns according to textual similarity. A Facility where several textual similarity metrics (e.g. character n-grams) is used to find similar tasks.
	perform Structured Task Search	Retrieval of Tasks/Cases/Patterns according to matching attributes
Task Patterns	apply Pattern On Current Task	
General Task Handling	Move Sub-task To Position X	
	Get Related Task (allowed Dependency types)	
	Formal Concept Recommendation	For certain attributes it is foreseen that the user can use a simple textual description as well as a more formal concept (class, instance). By means of information extraction, the user should get suggestions of formal concepts according to the entered textual description.
Search for Experts / Executors	Find Person with Ability X	Find Person according to Abilities – e.g. Skills
	find Person for Timeframe X	Find Person according to who has unoccupied time frames

Table 6.8: High level list of functionality which is supposed to be relevant on realizing the system.

6.3.4. Summary of Nepomuk Task

In the previous sections, we have shown how we designed the central class of the Task Model Ontology for the SSD. Figure 6.14 gives the complete picture of this class including all direct attributes. Figure 6.15 shows all attributes, which have classes modelled within the TMO as type.

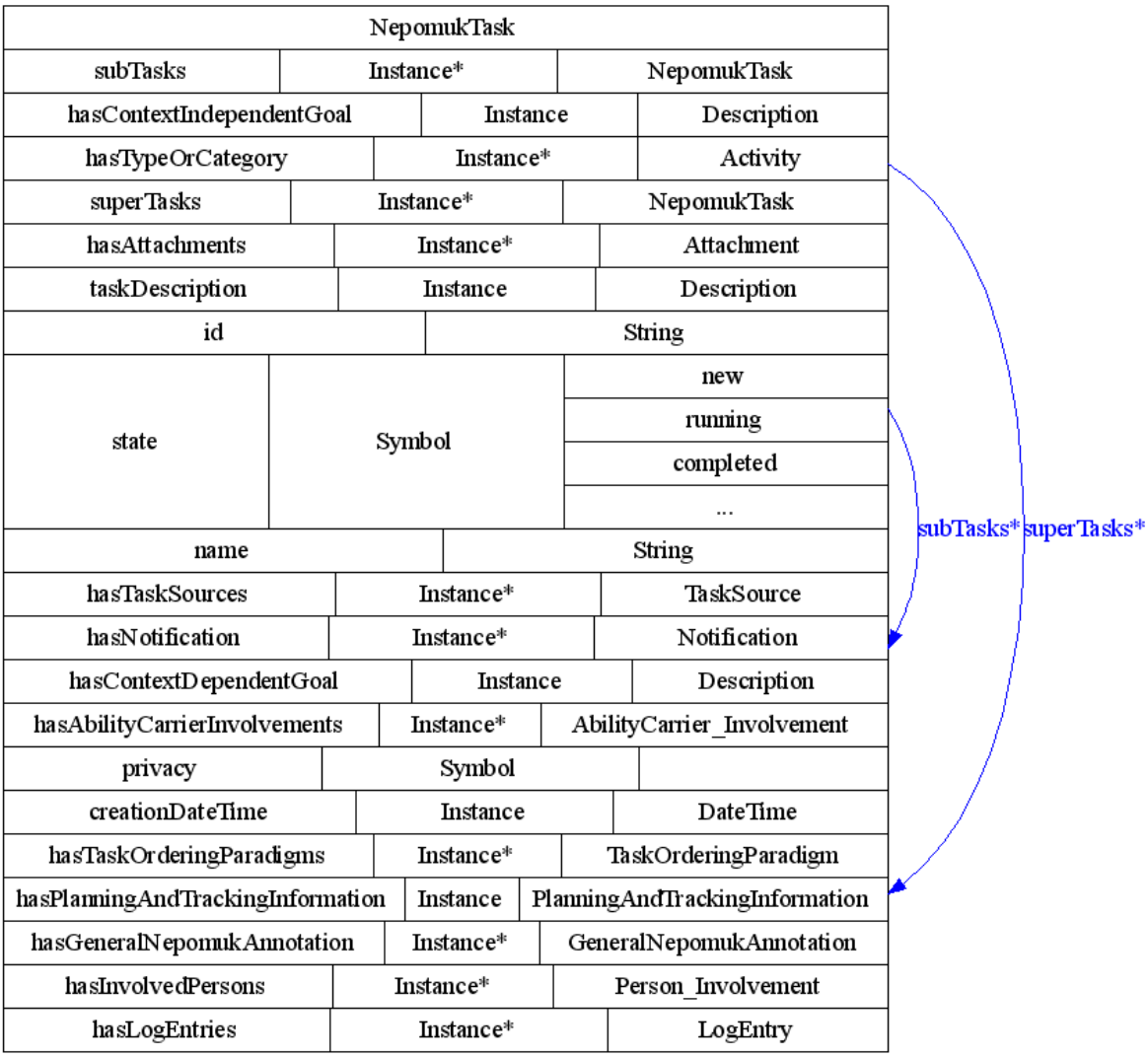

Figure 6.14: The NepomukTask Class with all direct attributes.

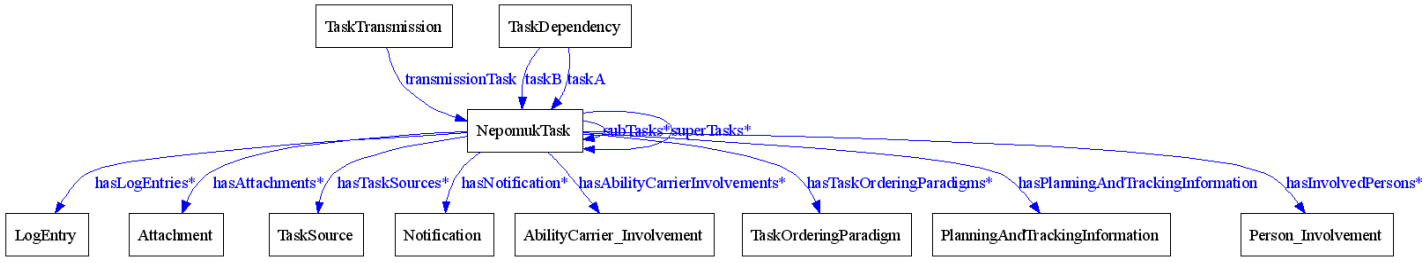

Figure 6.15: The NepomukTask Class with all "non-primitive" attributes.

6.4. Task Patterns

Task patterns, as described in Section 5.2.2, aim for the reuse of prior knowledge. In the following, we will describe the two main directions in which task patterns are treated: They are created by an abstraction process and task instances are created from patterns in an instantiation process.

6.4.1. Task and Pattern Repository (TPR)

The **T**ask and **P**attern **R**epository (TPR) is a facility where cases (used task instances) and task patterns (abstracted task instances) are stored for reuse. On the TPR, it is possible to perform the search for stored task knowledge. TPR's are the place where exchange of reusable task knowledge between socially connected individuals occurs.

We separate between personal and organizational TPR's. Whereas a personal TPR exists on the Social Semantic Desktop, an organizational TPR exists within a organization and is used by several Social Semantic Desktops. The personal TPR is the archive of a PTM. In an organizational TPR patterns and cases from several personal TPR's are merged together. For organizational TPR there will likely be maintenance and consolidation of task knowledge by voluntary or obliged humans. An organizational TPR allows creating patterns based on several similar cases.

6.4.2. Task Pattern Model and Lifecycle

6.4.2.1 Task Case Abstraction

Task case abstraction, resulting in the creation or update of task patterns, emanates from existing task cases. This typically takes place in the personal Task and Pattern Repository. This is in general a difficult and effort-intensive process, starting with the identification of selected task attributes for generalisation. These are typically context-dependent attributes such as task name but may extend to other attributes identified by the system as deviations from the underlying task pattern (if relevant). The user then proceeds to remove case-specific details and generalise these in a fashion that is more amenable to reuse in similar task situations. Finally, the abstracted task case is created or updated in the pattern repository.

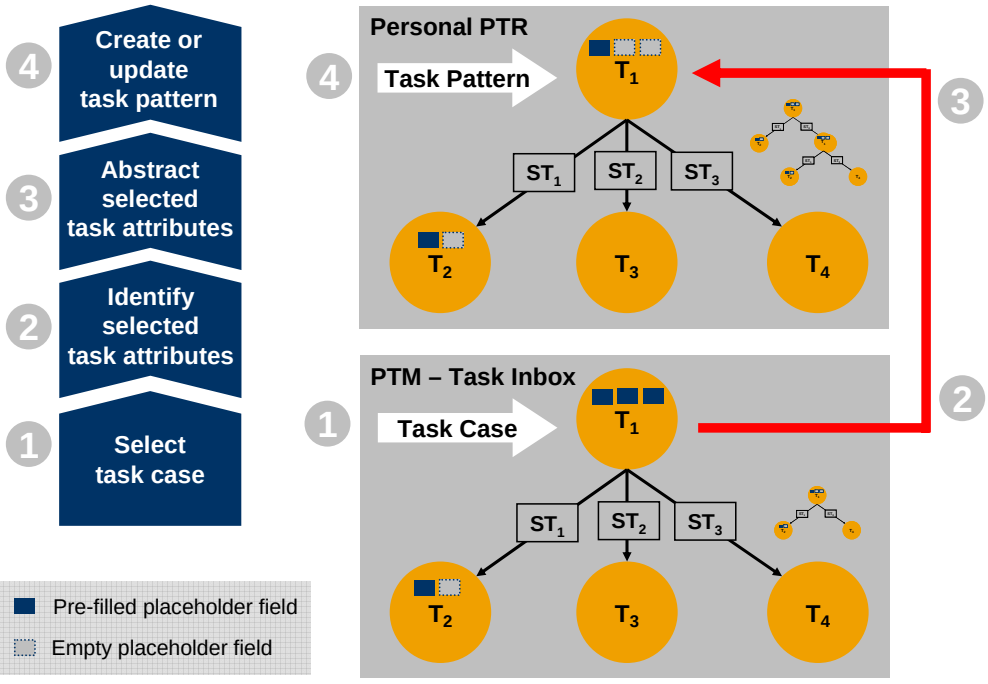

Figure 6.15: Task pattern lifecycle – task case abstraction.

6.4.2.2 Task Instantiation

Task instantiation from task patterns is more or less a reverse of the task case abstraction process. The user searches or browses the pattern repository for suitable task patterns matching the characteristics of the task at hand. In the event that specialised task patterns are available, e.g. travel planning task pattern vs. travel-to-Karlsruhe planning task pattern, the more general task pattern is recommended. However, all matching task patterns are displayed and ranked by relevance. The user may use the recommended task pattern as a search template to refine his search until he selects the task pattern from which he wishes to instantiate. Note that the new task instance may be embedded as a sub-task within a super-task or may be a top-level task within his PTM.

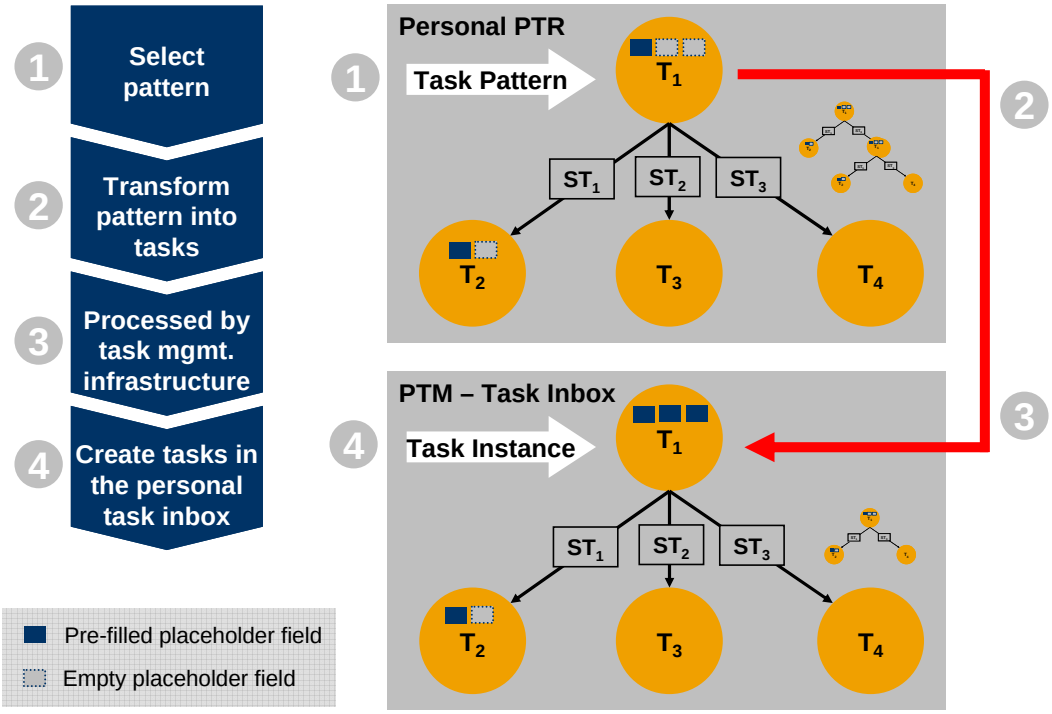

Figure 6.16: Task pattern lifecycle – task instantiation.

6.4.2.3 Statistical attributes

Task patterns have additional requirements beyond those of tasks. Where the aim of tasks is to support the management and execution of work activities, the aim of task patterns is to support the management and transfer of work experience across tasks and social contexts. To this end, additional attributes are necessary to record, say, statistical usage information of tasks derived from a given task pattern, and selected aspects of the task history such as sub-task reorganisation and execution times.

The following table describes some additional attributes specific to task patterns. However, this should be open to extension and modification as new insights are gained.

Attribute	Description
Usage Count `[usageCount:Integer]0:1`	The number of times the task pattern has been applied.
Total usage duration `[totalUsageDuration:Integer]0:1`	The total duration across all tasks instantiated from the task pattern.
Sub-task deletion `[deletedSubtasks:NepomukTask]0:*`	A list of sub-tasks deleted from task instances derived from the task pattern during task execution.
Sub-task reorganisation `[reorganizedSubtasks:NepomukTask]0:*`	A list of sub-tasks reorganized within task instances during task execution e.g. change of sequence, moved into sub-task.
Update Count `[update:Integer]0:1`	The number of times the task pattern has been revised.

Table 6.9 – Task pattern specific attributes.

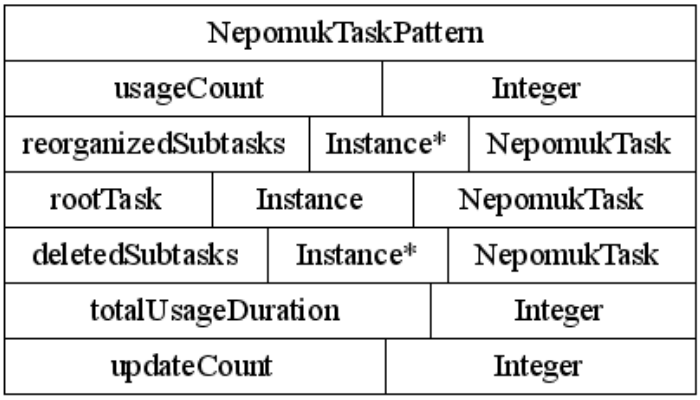

Figure 6.17: Task pattern.

7. Conclusion

Theories such as Activity Theory and Coordination Theory provide a powerful basis for the development of the Nepomuk Task Management. These theories do not provide direct information on the features that we have to consider for implementation, but they point at the aspects of the task management that are important. For example, Activity Theory described the constituents of the task such as actor, goal, and means, while Coordination Theory points at the necessity to describe tasks in their substructures, which might again consist of finer tasks related to parts of the work to be done.

The requirements that have been formulated in the case studies are not sufficient to define the task management model. They can only point at necessary conditions that we must fulfil with the task management. Therefore, the requirements played only a minor role in the document. We only used it to control that our task model provides the individually formulated requirements.

The conceptual task model provides an integration of the collected ideas in a consistent way, however, on a yet informal level. Here all relevant aspects from the different theories and exiting models are combined to one concept that is the basis for the formal ontological description that is provided at the end.

The ontological representation of the task management is a central gateway to the semantic work of Nepomuk and allows us to seamless integrate it into the provided semantic framework. Thus the task management becomes accessible to Nepomuk services and can seamless contribute information to other Nepomuk components.

The representation of the task model in this report is only the first step towards a complete description. It mainly focuses on the structural components of the task model, describing corresponding classes and attributes, and masks out the dynamic aspects. Nevertheless, Section 5 considers the task functionality but only against the requirement to determine the structure in such a way that this functionality can be realized on this basis. This means that the functional description does not fully grasp all dynamic aspects.

Another issue that was not considered thoroughly as well are the topics of security and privacy. Naturally, these aspects play a central role in the task management but on the other hand, security and privacy issues cannot be solved at the level of task management alone. Therefore, we have postponed these issues to a general Nepomuk discussion, which is starting to evolve.

Summing up we consider the task model in the present form as a solid basis for the further development. This concerns the integration of the ontological structure in the global Nepomuk structure, e.g., with respect to PIMO. One of the goals of this report also was the identification of open questions and some of these could be identified. They will become the focus of the following discussions and the ongoing work.

8. References

8.1. State of the art analysis - task modelling

Blackler, F. (1995). Knowledge, Knowledge Work and Organizations: An Overview and Interpretation. Organization Studies, 16(6), 1021-1046.

Brunzel et al. (2006a) Marko Brunzel, Myra Spiliopoulou. Discovering Semantic Sibling Groups from Web Documents with XTREEM-SG. In Proc. of EKAW 2006, LNAI 4248, Podebrady, Czech Republic, October 2006.

Brunzel et al. (2006b) Marko Brunzel, Myra Spiliopoulou. Discovering Semantic Sibling Associations from Web Documents with XTREEM-SP. In Proc. of DAWAK 2006, LNCS 4081, Krakow, Poland, September 2006.

Brunzel et al. (2007) Marko Brunzel, Andreas Dengel, Myra Spiliopoulou. Indexing the WWW for Sibling Terms. submitted to WWW 2007.

Crowston et al. (2004) Kevin Crowston, Joseph Rubleske and James Howison. Coordination Theory: A Ten-Year Retrospective. Draft of 23 September 2004. To appear In Zhang, P. and Galletta, D. (Eds.) Human-Computer Interaction in Management Information Systems, M. E. Sharpe, Inc. http://crowston.syr.edu/papers/coord2004.pdf (Last accessed: 01/08/2007).

DFKI (2003). DFKI GmbH. Weak Workflows in FRODO TaskMan – System Walkthrough and Evaluation. 2003. http://www.dfki.de/frodo/taskman/taskman_eval_june03.pdf (Last accessed: 01/08/2007).

Dourish (1996) Dourish, P., Holmes, J., MacLean, A., Marqvardsen, P., and Zbyslaw, A.: Freeflow: mediating between representation and action in workflow systems. In *Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work* (Boston, Massachusetts, United States, November 16 - 20, 1996). M. S. Ackerman, Ed. CSCW '96. ACM Press, New York, NY, 190-198.

Dustdar (2004) Dustdar, S.: Caramba—A Process-Aware Collaboration System Supporting Ad hoc and Collaborative Processes in Virtual Teams. *Distrib. Parallel Databases* 15, 1 (Jan. 2004), 45-66.

FRODO (2003). Andreas Abecker, Ansgar Bernardi, Ludger van Elst and Andreas Lauer, Heiko Maus, Michael Sintek and Sven Schwarz. FRODO: ("A Framework for Distributed Organizational Memories"). Abschlußbericht}, BMBF Abschlussbericht, FKZ 01 IW 901. 2003.

FRODO (2005) FRODO ("A Framework for Distributed Organizational Memories") Website, German Research Center for Artificial Intelligence, http://www.dfki.uni-kl.de/KM//content/e179/e506/index_eng.html (Last accessed: 01/08/2007).

Gollwitzer (1990) Gollwitzer, P. M.: Action Phases and Mind-sets. In: E. T. Higgins; R. M. Sorrentino: Handbook of Motivation and Cognition, Vol. 2, Guilford, New York, 1990.

Grebner (2006) Grebner, Olaf. Service-oriented task management. Diploma thesis, Department of Business Information Systems, Darmstadt University of Technology, Germany 2006.

Guareis de Farias et al. (2000) Guareis de Farias, C.R., Ferreira Pires, L. and van Sinderen, M. 2000: A conceptual model for the development of CSCW systems. Fourth International Conference on the Design of Cooperative Systems. To appear.

Jonassen, D.H.; Murphy, M. (1998). Activity theory as a framework for designing constructivist learning environments. St. Louis, Missouri: Paper presented at the annual meeting of the Association for Educational Communications and Technology.

Kreifelts et al. (1993) Thomas Kreifelts, Elke Hinrichs, Gerd Woetzel (Gesellschaft für Mathematik und Datenverarbeitung, Germany) Sharing To-Do Lists with a Distributed Task Manager. ECSCW '93, Proc. Third European Conference on Computer-Supported Cooperative Work, September 15-17, 1993, Milano, Italy.

Kuutti (1991) Kuutti, K.: The concept of activity as a basic unit of analysis for CSCW research, Proc. of the European Conf. on Computer Supported Cooperative Work ECSCW'91.

Leontiev (1978). Leontiev, A. N: Activity, Consciousness, and Personality. Eaglewood Cliffs, NJ, Prentice-Hall, 1978. (Original work published in Russian in 1975).

Leontiev (1981). Leontiev, A. N.: Problems of the development of the mind. Moscow, Progress, 1981.

Malone Crowston (1990) Malone, T. W. and Crowston, K. 1990. What is Coordination Theory and How Can It Help Design Cooperative Work Systems? Proceedings of the 1990 ACM conference on Computer-supported cooperative work, Los Angeles, California, United States, pp. 357 – 370.

Malone Crowston (1994) Malone, T. W. and Crowston, K. 1994. The interdisciplinary study of coordination. *ACM Comput. Surv.* 26, 1 (Mar. 1994), 87-119. DOI= http://doi.acm.org/10.1145/174666.174668 (Last accessed: 01/08/2007).

Medina-Mora et al. (1992). Medina-Mora, R. et al,: 1992, The action workflow approach to workflow management technology, Proc. of the Conf. on Computer Supported Cooperative Work CSCW92.

Moran (2005). Thomas P. Moran. "Unified Activity Management: Explicitly Representing Activity in Work-Support Systems.", *Proceedings of the European Conference on Computer-Supported Cooperative Work (ECSCW 2005), Workshop on Activity: From Theoretical to a Computational Construct* (2005).

Moran (2005a) Thomas P. Moran. Unified Activity Management for BPDM. 2005/7/22. www.bpmn.org/Documents/BPDM/2005-07-22%20uam%20BPDM.ppt (Last accessed: 01/01/2006).

Moran et al. (2005) Moran, T.P., Cozzi, A., Farrell, S.P. (2005). Unified Activity Management: Supporting people in e-business. Communications of the ACM, 48(12).

Moran et al. (2006). Moody, P.; Gruen, D.; Muller, M.J.; Tang, J.; Moran, T.P. (2006). Business activity patterns: A new model for collaborative business applications. IBM Systems Journal, Vol. 43, No. 4, 683-694.

Polanyi (1966). Polanyi, M.E. (1966). Personal Knowledge: Towards a Post-Critical Philosophy, University of Chicago Press, Chicago, IL.

Prinz (1994) Prinz, Wolfgang, "Object-oriented organization modeling for the support of CSCW," System Sciences, 1994. Vol.IV: Information Systems: Collaboration Technology Organizational Systems and Technology, Proceedings of the Twenty-Seventh Hawaii International Conference on , vol.4, no.pp.797-806, 4-7 Jan 1994. URL: http://ieeexplore.ieee.org/iel2/934/7712/00323429.pdf?isnumber=7712∏=STD&arnumber=323429&arnumber=323429&arSt=797&ared=806&arAuthor=Prinz%2C+W (Last accessed: 01/08/2007).

Riss et al. (2005) Riss, U.V.; Rickayzen, A.; Maus, H.; Aalst, W. M. P. v. d.: Challenges for Business Process and Task Management. J.UKM 0 (2), (2005) 77-100.

Riss et al. (2006) Riss, U. V.; Cress, U.; Kimmerle, J.; Martin, S. (2006) Knowledge Transfer by Sharing Task Patterns - From Experiment to Application. Proceeding of KMAC 2006, Birmingham.

Riss et al. (2007) Riss, U.V.; Jarodzka, H.M.; Grebner, O.: Pattern-based task management and implicit knowledge. NAIK Workshop at the 4th Conference on Professional Knowledge Management, Potsdam, Germany.

Teege (1996) Gunnar Teege, Object-Oriented Activity Support: A Model for Integrated CSCW Systems, Computer Supported Cooperative Work: The Journal of Collaborative Computing 5, pp.93-124, 1996.

Winograd (1988) Winograd, T.: 1988, A Language/Action Perspective on the Design of Cooperative Work, Human Computer Interaction 3(1), 3 30.

8.2. Task Management Model Requirements – Scoping

Nepomuk D8.1 (2006) Nepomuk Consortium. Deliverable D8.1: Institute Pasteur Scenario Report. http://nepomuk.semanticdesktop.org/xwiki/bin/download/IST/WebHome/D8.1_v31_NEPOMUK_IP_Scenario_Report.pdf (Last accessed: 01/08/2007).

Nepomuk D9.1 (2006) Nepomuk Consortium. Deliverable D9.1: TMI Scenario Report. http://nepomuk.semanticdesktop.org/xwiki/bin/download/IST/WebHome/D9.1_v32_NEPOMUK_TMI_Scenario_Report.pdf (Last accessed: 01/08/2007).

Nepomuk D10.1 (2006) Nepomuk Consortium. Deliverable D10.1: SAP Scenario Report. http://nepomuk.semanticdesktop.org/xwiki/bin/download/IST/WebHome/D10.1_v11_NEPOMUK_SAP_Scenario_Report.pdf (Last accessed: 01/08/2007).

Nepomuk D11.1 (2006) Nepomuk Consortium. Deliverable D11.1: Mandrake Community Scenario Report. http://nepomuk.semanticdesktop.org/xwiki/bin/download/IST/WebHome/D11.1_v11_NEPOMUK_Mandriva_Community_Scenario_Report.pdf (Last accessed: 01/08/2007).

8.3. Conceptual Task Management Model

WfMC (1999) Workflow Management Coalition. Terminology & Glossary. Document Number WFMC-TC-1011, Document Status - Issue 3.0. Feb 99. http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf (Last accessed: 01/08/2007).

8.4. Task Management Model

NRL (2006) NEPOMUK Representational Language (NRL) Vocabulary Specification [Draft] http://svn.nepomuk.semanticdesktop.org/repos/trunk/taskforce/TF-Ont/draft/NRL.html (Last accessed: 01/08/2007).

PIMO Ontology (2006) http://nepomuk.semanticdesktop.org/xwiki/bin/view/Main/PimoOntologyDFKI (Last accessed: 01/08/2007).

PIMOS (2006) NEPOMUK pimos - the Personal Information Model Structures http://nepomuk.semanticdesktop.org/xwiki/bin/view/Main/pimos (Last accessed: 01/08/2007).

Sauermann (2006) Leo Sauermann. PIMO - a PIM Ontology for the Semantic Desktop. http://www.dfki.uni-kl.de/~sauermann/2006/01-pimo-report/pimOntologyLanguageReport.html (Last accessed: 01/08/2007).

Sintek (2001) Michael Sintek. OntoViz Protégé Plug-In. http://protege.cim3.net/cgi-bin/wiki.pl?OntoViz (Last accessed: 01/08/2007).

Sintek et al. (2006) Michael Sintek, Ludger van Elst, Simon Scerri, Siegfried Handschuh. Distributed Knowledge Representation on the Social Semantic Desktop: Named Graphs, Views and Roles in NRL. Submitted to European Semantic Web Conference 2007.

Spyns et al.(2002) Spyns P., Meersman R. & Jarrar M., Data modelling versus Ontology engineering. SIGMOD Record: Special Issue on Semantic Web and Data Management, 31(4) : 12 - 17, December 2002.

TF-Ont (2006) NEPOMUK Ontologies Task Force http://nepomuk.semanticdesktop.org/xwiki/bin/view/Main/TF-Ont (Last accessed: 01/08/2007).

TF-PIMO (2006) NEPOMUK Personal Information Management Ontology Task Force http://nepomuk.semanticdesktop.org/xwiki/bin/view/Main/TF-PIMO (Last accessed: 01/08/2007).