Towards a Formalization of Individual Work Execution at Computer Workplaces

Benedikt Schmidt[1], Heiko Paulheim[1], Todor Stoitsev[1], and Max Mühlhäuser[2]

[1] SAP Research Darmstadt,
Bleichstrasse 8, 64285 Darmstadt, Germany
{firstname.lastname}@sap.com
[2] Technische Universität Darmstadt, Telecooperation Group
Schloßgartenstr. 7, 64289 Darmstadt, Germany
{firstname}@informatik.tu-darmstadt.de,

Abstract. To better understand, analyze, and support work execution at computer workplaces, this paper presents a framework of ontologies. We analyze knowledge work at computer workplaces as weakly-structured processes by means of activity theory. Based on the analysis, we extend a set of upper ontologies to model the computer workplace and the process of work execution. We especially reflect the process of tool selection involved in work execution by a hierarchical analysis of involved planning activities and software tools, enabling a plan realization.

1 Introduction

Individual work execution processes in knowledge work are complex and weakly structured, i.e., they allow a large number of variations of the individual process steps and their execution order. A relevant environment of individual work execution processes is the computer workplace. This paper addresses the formalization of work execution at computer workplaces, i.e. the execution of regular office work.

Computer workplaces are multi-purpose workplaces, providing a set of software applications with numerous functionalities to enable and support a large variety of information creation and consumption activities. Individuals blend the use of software applications in individual processes of work execution which manifest expertise as well as experience. Description, analysis and support of such individual execution processes is a difficult task, as it comprises consideration of the software landscape with its capabilities – a highly structured domain – as well as the planning and execution of the work process – a weakly structured and highly individualized process.

The remainder of this paper is structured as follows. We give an overview of the requirements for an ontology of the domain and introduce a running example that will illustrate our work throughout the paper. The third section analyzes work execution processes at computer workplaces in terms of activity theory (AT). To formalize the processes, we extent the DOLCE upper ontology that

is introduced in Sect. 4. The original contribution of this paper, the computer work ontology (CWO) is presented in Sect. 5:

- Formalization of the computer workplace
- Formalization of work execution processes and tool selection

Finally, we review related ontologies and conclude with a summary and an outlook on future work.

2 Running Example and Requirements

Although the approach supports the formalization of arbitrary work execution processes at computer workplaces, we provide one running example that illustrates the core aspects of our work and that is used throughout the paper to illustrate formalizations.

2.1 Running Example: Pete Plans a Conference Travel

Pete works at the research department of a software company. Pete wants to visit the ICCS11 conference. Pete's original motive of visiting the conference is a strong interest in conceptual structures. Therefore, he has the objective to attend the ICCS11, which requires the creation of a travel request to be confirmed by his manager. The travel request is a standardized form which requires different information, e.g. travel destination, travel duration, and approximate costs for flight and hotel. Pete needs to find the travel request form, identify all required information, fill them into the form, and provide his manager with the filled out form. All these activities are performed on a windows PC with an office suite. To execute the task, Pete executes the following process:

- Browse for the document $\overset{solvedby}{\rightarrow}$ Use the Windows Explorer to identify the form document by opening different folders (Step1).
- Consume the authoring form document $\overset{solvedby}{\rightarrow}$ Open the form document with Microsoft Excel, as it is an .xls, file and identify required information (Step2).
- Browse for required information, conference data $\overset{solvedby}{\rightarrow}$ Open a web browser and use a search engine to access the conference website (Step3).
- Author form document, conference data $\overset{solvedby}{\rightarrow}$ Use copy and paste to transfer different information from the conference website to the form (Step4).
- Browse for required information, hotel and flight costs $\overset{solvedby}{\rightarrow}$ Use a web browser to access web pages to book flights and hotels and identify approximate costs (Step5).
- Author form document, hotel and flight costs $\overset{solvedby}{\rightarrow}$ Type identified costs into the respective input fields of the form document (Step6).
- Communicate the filled out form $\overset{solvedby}{\rightarrow}$ Start Outlook, create an email and attach the filled out request form (Step7).

In the remainder of this paper, we will use this example process to illustrate our concepts and illustrate the ontology. Since we are dealing with weakly structured processes, however, we have to point out that this only one out of the numerous valid execution paths which exist due to individual requirements and selections of available functionality.

2.2 Modeling Requirements

The example illustrates a work execution process at the computer workplace and gives us some indications of what concepts a useful formal description may contain. The process of individual requirement generation and the selection of appropriate software to solve the requirements is of importance. This will facilitate the modeling of execution processes and may also be used to facilitate work execution by improved user support: proposing software functionalities and resources.

Altogether, the example indicates the following required aspects:

1. Individual work execution: Information about individuals, individual goals and the categories which are used to describe and execute work is required.
2. Tool/artifact: Software and its capabilities on an abstract and a functional level needs to be modeled. Information handling needs to be a specific focus with respect to the structured encoding systems and the content.
3. Connection between individual execution and tools/artifacts: As the individual execution process is constrained by usable tools and artefacts, the connection between both needs to be modeled.

Following [11], we define four additional requirements for the formalization: We need to (1) avoid conceptual ambiguity, (2) axiomatize our concepts, (3) avoid concepts that have no ontological meaning but exist for modeling reasons only and (4) provide the concepts without limiting future extensions of the ontology.

3 Activity Theory as Perspective on Work Execution

Work execution at computer workplaces can be explained by means of Activity Theory (AT) [7]. AT provides concepts to describe the interaction of an individual with the environment. In the following, we give a quick overview on concepts of AT which are relevant in the context of this paper. Then, we apply these concepts to the domain of work execution at the computer workplace.

3.1 Activity Theory and Situated Behavior

AT uses activities to provide a minimal meaningful context for human action. Thereby, an activity is a form of doing directed to an object. The relation of a subject to an object is mediated by a tool. The tool condenses the historical development of the relationship between subject and object. The role of tools

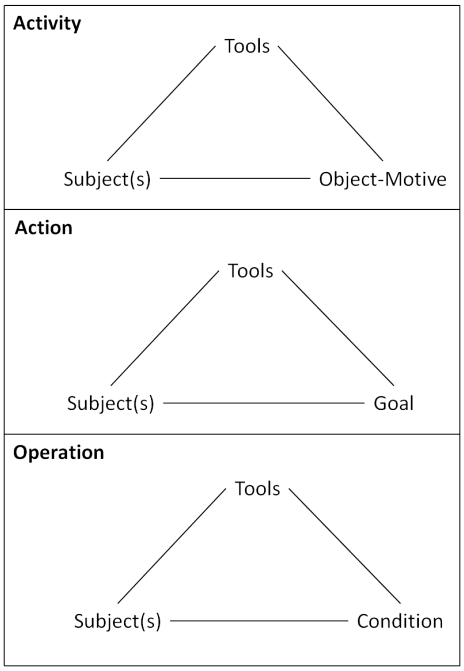

Fig. 1. Hierarchy of Behavior Situatedness in Activity Theory

shows the need to consider artifacts as integral and inseparable components of human functioning.

Based on the activity, AT organizes situated behavior in a hierarchy, decomposing activities to actions and operations. Each element of the hierarchy is a maximally connected triad (see Fig. 1). Activities as long-term formations cannot be transformed directly into outcomes, but through a process. Thereby, activities are realized by a set of individual and cooperative actions. These actions are related to each other by the object/motive of the enclosing activity. Actions are executed by a set of operations, which are routines used subconsciously as answers to conditions faced during the performance of the action [6].

The borders between the levels of the hierarchy are permeable. Actions transform to operations, once the subject has gained experience to execute the action subconsciously. In case of changing conditions, the operation may transform back into an action. An activity can become an action, once a motive is lost, or the goal of an action may turn out as being a motive, transforming the action to an activity.

3.2 Situated Behavior at Computer Workplaces

Considering the computer workplace, situated behavior presents is constrained by the capabilities of the computer as tool. The computer transforms signs and has established as a tool for supporting the consumption, creation and transformation of data as information. Precisely, all behavior is channeled through software tools with respective functionalities.

As described above, AT mediates activities, actions and operations by tools. Work at a computer workplace as an individual and weakly structured process involving different applications to generate a specific outcome can be considered as action. We do not consider weakly structured computer work as an activity, as the limited execution time and the precise outcome does not fit the long-term motive perspective. On the other hand, we do not consider weakly structured computer work as an operation, as the execution is a knowledge-intensive act with characteristics of problem-solving, thus improbably unconsciously executed.

Considering weakly structured work at computer workplaces as an action does not provide information about the way execution processes develop. In the given example (see Sect. 2), the main goal of requesting a travel by the manager was decomposed into different subgoals. Each subgoal was executed by the functionality of a software application. We propose the decomposition of computer work actions into four categories that are connected in the sense of a hierarchical decomposition:

- ***Task process:*** The first level in a computer work execution hierarchy is the process of planning and executing a task. We follow a functional understanding of task as a a logical unit of work that is performed by a series of actions in pursuit of a certain aim [8, 15].
- ***Knowledge action:*** The second level in the hierarchy stands for the initial decomposition of the task. The subject identifies the main challenges of the task and tries to address these challenges by patterns of problem solving. We call these problem solving patterns *knowledge actions*, following existing works on reoccurring problem solving tasks in knowledge work [5]. Relevant knowledge actions in the domain of computer workplaces are 1) Browsing, 2) Consuming, 3) Authoring, 4) Communicating, and 5) Organizing.
- ***Application action:*** The third level in the hierarchy stands for execution activities in the context of a software and the identification of sets of functionalities that need to be performed to execute the knowledge action.
- ***Desktop operation:*** The fourth and lowest level of a computer work execution hierarchy stands for single functionalities that can be accessed within the context of a software and thus can be realized directly by an operation.

4 Modeling Basis

While domain ontologies focus on a minimal terminological structure, upper ontologies describe general concepts which are valid across all knowledge domains. The ontology we present in this paper is an extension of the DOLCE ontology.

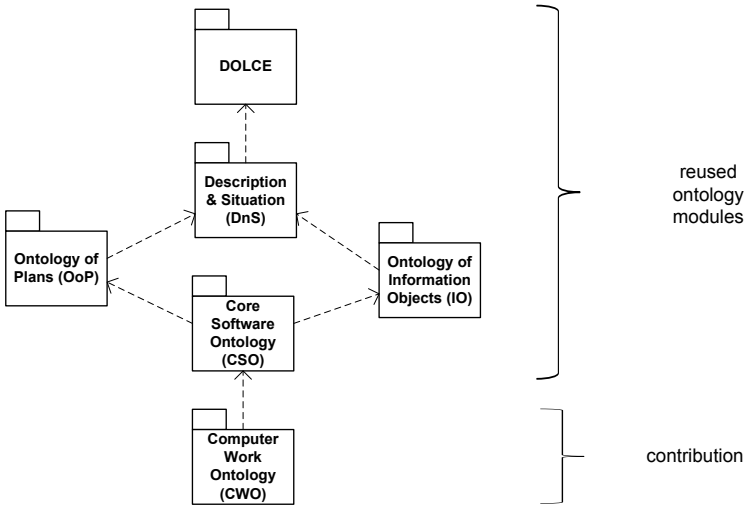

Fig. 2. Overview of the ontologies. Dotted lines represent dependencies between ontologies. An ontology O_1 depends on O_2 it if specializes concepts of O_2, has associations with domains and ranges to O_2 or reuses its axioms.

DOLCE has been designed as a first module of a foundational Ontologies Library [3, 9]. As an upper ontology, DOLCE describes relationships between endurant and perdurant particulars. Endurants are independent essential wholes that are in time, while lacking temporal parts, e.g. this paper. Perdurants, on the contrary, are entities that happen in time, and can have temporal parts, e.g. the process of reading this paper.

The library of ontologies contains different systematically related modules and defines different design patterns to reuse the content for more specific domains [2]. For this paper, we reuse the following ontologies (c.f. Fig. 2):

- Descriptions and Situations (DnS): An ontological theory of contexts. DnS can be considered an ontology design pattern for structuring core and domain ontologies that require contextualization.
- Ontology of Plans (OoP): Formalization of a generic theory of plans.
- Ontology of Information Objects (IO): A semiotic ontology design pattern that assumes a content transferred in any modality to be equivalent to a social object called information object.
- Core Software Ontology (CSO): Formalization of fundamental concepts in the computer domain, e.g. software or data.

5 Computer Work Ontology (CWO)

The CWO extends the DOLCE ontology with respect to the domain of individual, weakly-structured desktop work. We present the CWO in terms of tool,

subject, and object and describe their relationship as given in the AT triade. The tool is the computer workplace environment, which is described as an environment for the transformation of information by functionalities. The subject is the knowledge worker, who plans and executes work at a computer workplace. The object is the goal of the individual work.

5.1 Tool: Computer Workplace Environment

We model the computer workplace as an environment that offers functionalities of generating, displaying and transforming data which can be consumed as information. The functionalities and the available information defines a possibility-space for the execution of work. Functionalities are encapsulated in software tools and information is stored in files.

Software and Functionalities To model software, we use the respective design pattern as described in [11]: CSO:Software[3] is defined as CSO:Data that OIO:expresses an OoP:Plan, itself sequencing a set of OoP:Task (see Fig. 3). We are interested in a perspective on software, as it is available to end-users. The functionalities offered by the software are modeled as CWO:Functionality, a specialization of OoP:Task. To describe the plans, describing the purpose-of-use of a software (e.g. word processing), we model CWO:Scenario as specialization of OoP:Abstract-Plan. The CWO:Scenario sequences a set of CWO:Functionality.

(D1) CSO:Functionality(x) $=_{def}$ OoP:BagTask(x)
$\wedge \exists$ y (DOLCE:part-of(y,x) $\wedge$ ComputationalTask(y))

(D2) Scenario(x) $=_{def}$ OoP:Abstract-Plan(x)
$\wedge \forall$ y (DnS:defines(x,y) $\rightarrow$ Functionality(y))

(D3) CSO:Application(x) $=_{def}$ CSO:Software(x)
$\wedge \exists$ y (OIO:realizedBy(x,y) $\wedge$ CSO:ComputationalObjects(y)
$\wedge \forall$ z (OIO:expresses(x,z) $\rightarrow$ Scenario(z))

As an example, we model the Windows Explorer functionality to open folders and display files, that was used to identify the form document. We relate a scenario with the software and assign different functionalities to the scenario.

(Ex1) CSO:Software(windowsExplorer)
(Ex2) Scenario(folderStructureInteraction)
(Ex3) Functionality(browseFolderStructure)
(Ex4) Functionality(getElementDetails)
(Ex5) Functionality(executeElementWithApplication)
(Ex6) OIO:express(windowsExplorer,folderStructureInteraction)
(Ex7) DnS:defines(folderStructureInteraction,browseFolderStructure)
(Ex8) DnS:defines(folderStructureInteraction,getElementDetails)
(Ex9) DnS:defines(folderStructureInteraction,
executeElementWithApplication)

[3] Throughout the paper entities that belong to CWO are given without prefix. For all other entities, the respective prefix is given.

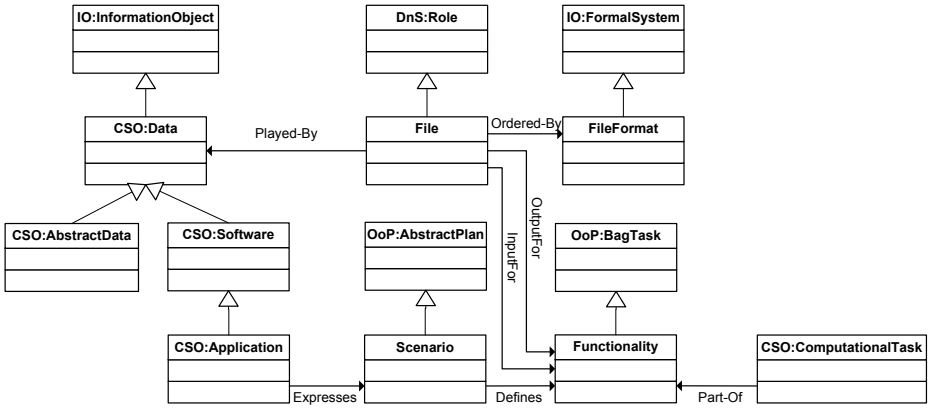

Fig. 3. The classification of software with scenarios, functionalities, and files. Concepts taken from DOLCE and accompanying ontologies are labeled with the respective name space.

Information Objects Represented by Files Files realize a connection between meaningful information and software by data in a digital encoded representation. We model a CWO:File as a role played-by only CSO:Data. As CSO:Software is a subclass of CSO:Data, we cover software as files (see Fig. 3). CSO:AbstractData is another subclass of CSO:Data, containing data that identifies something different from itself, e.g., the word *tree* that stands for a mental image of a real tree. As a file may be abstract data or software, two aspects of files are supported: 1) being a static information object 2) being an information object for execution to make plans accessible in a runtime representation. A file as a static information object is modeled by relating the file as CSO:Data by DnS:about with a DnS:description. A file as an executable information object relates CSO:Software with OoP:Plan by the DnS:expresses relation.

A CWO:File is DnS:ordered-by a CWO:File-Format. A CWO:File with specific CWO:File-Formats can be input for CWO:Functionality. This connection organizes the file access by functionalities, which may range from opening the file to display content in a work processor to the interpretation of a web page by a web browser.

(D4) File-Format(x) $\rightarrow$ IO:Formal-System(x)
(D5) specializes(x,y) $\wedge$ File-Format(x) $\rightarrow$ File-Format(y)
(D6) uses(x,y) $\wedge$ File-Format(x) $\rightarrow$ File-Format(y)
(D7) File(x) $=_{def}$ DnS:Role(x) $\wedge$ $\exists$y(ordered-by(x,y) $\wedge$ File-Format(y))
$\wedge$ $\exists$z(played-by(z,x) $\wedge$ (AbstractData(z) $\vee$ Software(z)))
$\wedge$ $\forall$f(inputFor(x,f)$\rightarrow$Functionality(f))
$\wedge$ $\forall$g(outputFor(x,g $\rightarrow$Functionality(g))

In the following, we give two examples for using CSO:File. The first example is for a file as role played by CSO:Data that is not CSO:Software. This means, the

aspect of being an information object is of prime importance. For this purpose we model the form document which is identified in step 1 of the example (see Sect. 2) and show the connection to a word processor.

(Ex10) IO:Information-Object(document-for-travel-request)
(Ex11) DnS:description(travel)
(Ex12) DnS:about(document-for-travel-request, travel)
(Ex13) File(TravelRequest.docx)
(Ex14) DnS:played-by(Document-for-travel-request, travelRequest.docx)
(Ex15) File-Format(docx)
(Ex16) DnS:ordered-by(travelRequest.docx, docx)
(Ex17) CSO:Software(microsoftWord)
(Ex18) Scenario(textProcessing)
(Ex19) DnS:expresses(microsoftWord,textProcessing)
(Ex20) Functionality(openTextFile)
(Ex21) DnS:defines(textProcessing, openTextFile)
(Ex22) DnS:inputFor(openTextFile, travelRequest.docx)

The second example is for a file as a role played-by CSO:software. This means that the file as software gives access to functionalities. An interesting example are web applications interpreted from the perspective of a user. For a user, a web application is an address to be typed into a browser. By focusing on this aspect of consumption, the web application is a software that plays the role of a file. We use step 5 of the example (see Sect. 2), which is opening and interacting with a web application to book hotels.

(Ex23) CSO:Software(hotelBooker)
(Ex24) Scenario(searchHotel)
(Ex25) DnS:expresses(hotelBooker,searchHotel)
(Ex26) File(www.hotelbooker.net)
(Ex27) File-Format(html4.0)
(Ex28) DnS:ordered-by(www.hotelbooker.net, html4.0)
(Ex29) DOLCE:played-by(www.hotelbooker.net, hotelBooker)
(Ex30) CSO:Software(firefox)
(Ex31) Scenario(webBrowsing)
(Ex32) DnS:expresses(firefox,webBrowsing)
(Ex33) Functionality(accessWebsite)
(Ex34) DnS:defines(webBrowsing, accessWebsite)
(Ex35) DnS:inputFor(openWebsite, www.hotelbooker.net)

5.2 Subject: Task Execution

Following our hierarchy for the action layer of AT (see Sect. 3.1), comprising task execution (CWO:TaskProcess), knowledge action (CWO:KnowledgeAction), application action (CWO:ApplicationAction), and desktop operation (CWO:Desktop-Operation), we apply the plan pattern of the OoP [2] (see Fig. 4). Modeling the task execution based on the OoP:AbstractPlan stresses the weak structure and adaptation of execution processes based on constraints we want to

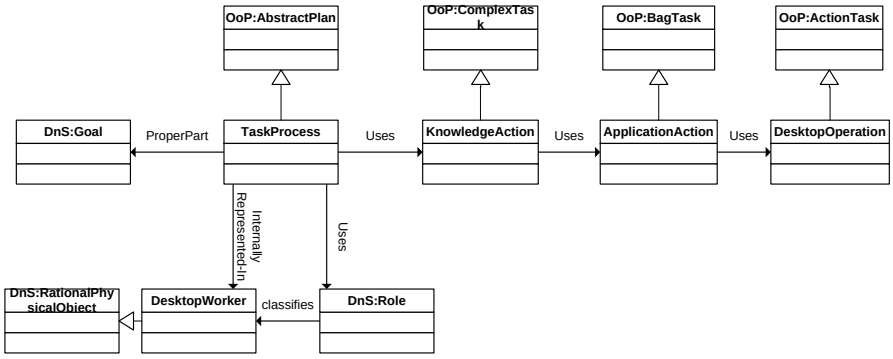

Fig. 4. The classification of the action hierarchy including TaskProcess, KnowledgeAction, ApplicationAction, and DesktopOperation and use of the planning pattern. Concepts taken from DOLCE and accompanying ontologies are labeled with the respective name space.

stress. An OoP:AbstractPlan describes methods for the execution of a procedure. A CWO:TaskProcess is internally-represented in an agent, has a goal, and uses at least one CWO:KnowledgeAction. Following a hierarchical model, each CWO:KnowledgeAction uses an CWO:ApplicationAction, which uses a CWO:DesktopOperation. A CWO:KnowledgeAction references a description it is about. An CWO:ApplicationAction references a CWO:SoftwareClass, which organizes software that shares similarities with respect to the tackled scenarios.

(D8) DesktopWorker(x) $=_{def}$ DnS:rational-physical-object(x)
$\wedge$ $\exists$ y(internally-represented-by(y,x) $\wedge$ TaskProcess(y))

(D9) SoftwareClass(x) $=_{def}$ DnS:Collection(x)
$\wedge$ $\forall$y(DnS:member(x,y)$\rightarrow$Software(y))

(D10) TaskProcess(x) $=_{def}$ OoP:AbstractPlan(x)
$\wedge$ $\forall$ y(ComplexTask(y) $\wedge$ uses(x,y) $\rightarrow$ KnowledgeAction(y))

(D11) KnowledgeAction(x) $=_{def}$ ComplexTask(x)
$\wedge$ $\forall$ y(uses(x,y) $\rightarrow$ ApplicationAction(y))
$\wedge$ $\exists$ z(references(x,z) $\wedge$ DnS:Description(z))

(D12) ApplicationAction(x) $=_{def}$ BagTask(x)
$\wedge$ $\forall$ y(uses(x,y) $\rightarrow$ DesktopOperation(y))
$\wedge$ $\exists$ z(references(x,z) $\wedge$ CSO:SoftwareClass(z))

(D13) DesktopOperation(x) $=_{def}$ ActionTask(x)
$\wedge$ $\exists$ y(uses(x,z) $\wedge$ Functionality(y))

To give an example, we show the respective decomposition for step 1 in the initial example of creating a travel request for the manager (see Sect. 2).

(Ex36) DesktopWorker(pete)

(Ex37) TaskProcess(createTravelRequest)

(Ex38) internally-represented-by(createTravelRequest,pete)

(Ex39) KnowledgeAction(browse)
(Ex40) description(travelRequest)
(Ex41) references(browse, travelRequest)
(Ex42) ApplicationAction(searchFile)
(Ex43) defines(browse,searchFile)
(Ex44) SoftwareClass(fileBrowser)
(Ex45) references(searchFile, fileBrowser)
(Ex46) DesktopOperation(openFolder)
(Ex47) defines(searchFile,openFolder)

5.3 Object: Work Execution

We have described the decomposition of work into a hierarchy of actions, sequenced by a plan. Actions in AT are mediated by tools. In the CWO, we have modeled tools as software expressing scenarios that define functionalities. The mediation by a tool includes a process of tool selection, as the subject identifies a tool that sufficiently supports a given goal. To model this mediation process, we introduce the CWO:sufficient-implementation relation as a specialization of DnS:intensionally-references. CWO:sufficient-implementation expresses that a OoP:task can be adequately executed by using a respective DOLCE:endurant. We use the CWO:sufficient-implementation to connect the CWO:KnowledgeAction and the CWO:DesktopOperation with software and functionality as tools, to model the possible space of work execution (see Fig. 5).

Although the mediation process is modeled, the actual execution of work is not represented in the ontology. Such a modeling would require the description of the actual perdurants carried out by the user, such as clicking with a mouse or typing with a keyboard [12]. Since our focus is rather on the abstract work processes themselves than their modality-dependent execution, we have not included that level of detail in the CWO.

(D14) KnowledgeAction(x) $=_{def}$ OoP:ComplexTask
$\wedge$ $\forall$y(sufficient-implementation(x,y)$\rightarrow$Scenario(y))
(D15) DesktopOperation(x) $=_{def}$ OoP:ActionTask
$\wedge$ $\forall$y(sufficient-implementation(x,y)$\rightarrow$ Functionality(y))

To illustrate the extension we again rely on the first step of the initial example (see Sect. 2). We connect the decomposition of the task to the different actions to the software chosen in the example.

(Ex48) TaskProcess(createTravelRequest)
(Ex49) KnowledgeAction(browse)
(Ex50) ApplicationAction(searchFile)
(Ex51) defines(browse,searchFile)
(Ex52) DesktopOperation(openFolder)
(Ex53) defines(searchFile,openFolder)
(Ex54) Software(windowsExplorer)
(Ex55) Scenario(folderStructureInteraction)
(Ex56) Functionality(browseFolderStructure)

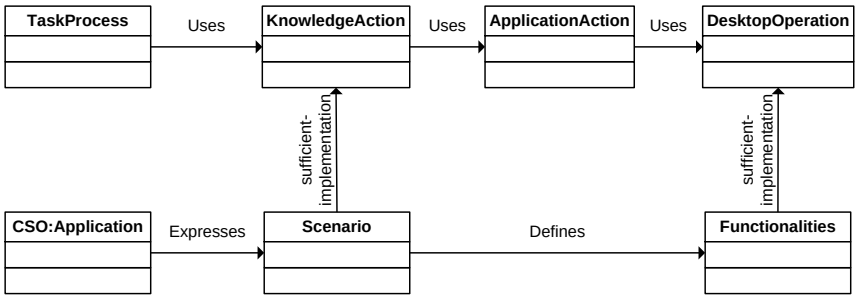

Fig. 5. The connection between the hierarchy of actions and software with scenarios and functionalities. Concepts taken from DOLCE and accompanying ontologies are labeled with the respective name space.

(Ex57) OIO:express(windowsExplorer,folderStructureInteraction)
(Ex58) defines(folderStructureInteraction,browseFolderStructure)
(Ex59) sufficient-implementation(searchFile,folderStructureInteraction)
(Ex60) sufficient-implementation(openFolder,browseFolderStructure)

6 Related Work

The work presented in this paper uses the DOLCE ontology and describes concepts to model the computer workplace and to model work execution processes as selection of appropriate tools for fixed goals. In the following we give an overview of related work in the domain of computer workplace modeling and execution process modeling.

Computer workplace modeling: Computer workplace modeling exists as information object modeling and as software application modeling. The personal information model (PIMO) ontology [14] and the Attention Meta Data [16] are two examples for formalizations of the existing information objects. Both approaches capture file types and apply methods of classification and categorization to organize files with respect to the content. Definitions of transformations based on the type of files are not captured by such ontologies, as the application landscape is out of scope.

Modeling applications may focus on application classes, using taxonomies of software applications [1]. Another focus is a standalone application and the interaction of the user with the application. [13] provides the UICO ontology that connects basic actions with resources and information needs. As UICO focuses on input for trained machine learning, a detailed model of applications is out of scope.

Execution process modeling: Formalizations of execution processes generally provide a vocabulary to specify goals and realize a sequential or hierar-

chical task decomposition. The decomposition is realized by elements like *Object* and *Activity* in [4] or *Goal* and *Act* in the "Act Formalism" [10]. Modeling of the domain and knowledge-intensive planning are not tackled in depth by the reviewed approaches. More formalizations of execution processes can be found in [2].

Overall related work: The focus on a single subject that organizes a personal task execution in the sense of an execution process is not in the focus of the described approaches. Especially, the integration of the computer workplace as domain and individual planning is not completely covered in any of the reviewed work. DOLCE with the DnS and the OoP extensions provides the necessary patterns, but requires additional classes to model the domain and additional properties to connect individual planning and the given domain, which has been realized by the CWO.

7 Conclusion

We have presented the Computer Work Ontology (CWO) that formalizes individual work execution in the domain of the computer workplace. Our ontology is grounded in foundational ontologies and enables the precise modeling of computer workplaces and a modeling of individual execution processes based on the definition of action in AT. We have shown the applicability of the CWO with a use case which was realized as running example in this paper. As we have applied modeling patterns belonging to DOLCE ontology, the CWO shows the applicability of those patterns for new domains.

The CWO focuses on a task perspective to describe work execution (endurants). As discussed above, user interactions like pressing a button on a keyboard, moving the mouse, etc. (perdurants) are out of scope for CWO. In the future, we are planning to connect our CWO with an already developed ontology of user interfaces and interactions [12]. As both ontologies are grounded the DOLCE ontologies, they are interoperable and can be integrated.

Currently, we use the CWO and DOLCE for two applications:

- *Task mining:* We enrich sensor events from a computer desktop with the respective data of the action hierarchy and create abstract patterns of work execution based on the captured work execution process.
- *Capturing document lifecycle:* We capture the transformation and dissemination of documents in a team of collaborative workers. The ontology captures the lifecycle of documents based on all acts of content enrichment, copying or disseminating.

In the future, we foresee to use CWO for providing proactive user support based on captured task instances. Thus, we will focus on the aspect of abstraction from specific work processes to more generic work processes.

Acknowledgements

The work presented in this paper has been partly funded by the German Federal Ministry of Education and Research under grant no. 01IA08006.

References

1. A. Forward and T. C. Lethbridge. A Taxonomy of Software Types to Facilitate Search and Evidence-Based Software Engineering. In *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, pages 1–13, 2008.
2. A. Gangemi, S. Borgo, and C. Catenacci. Task taxonomies for knowledge content. *METOKIS Deliverable*, 2004.
3. A. Gangemi, N. Guarino, and C. Masolo. *Sweetening ontologies with DOLCE*, pages 223–233. Springer, 2002.
4. M. Grüninger and C. Menzel. Specification Language (PSL) Theory and Applications. *AI Magazine*, 24(3):63–74, 2003.
5. T. Hädrich. Situation-oriented Provision of Knowledge Services. *Dissertation, Martin Luther Universität Halle-Wittenberg*, 2008.
6. K. Kuutti. *Activity theory as a potential framework for human-computer interaction research*, pages 17–44. Massachusetts Institute of Technology, Cambridge, MA, USA, 1995.
7. A. N. Leontiev. *Activity and consciousness*. Progress Publishers, 1977.
8. G. Marchionini. *Information Seeking in Electronic Environments*. Cambridge University Press, 1995.
9. C. Masolo, S. Borgo, A. Gangemi, N. Guarino, A. Oltramari, and I. Horrocks. WonderWeb Deliverable D18 Ontology Library (final) WonderWeb Project . *Communities*, 2001.
10. K. Myers and D. Wilkins. The Act Formalism, Version 2.2. *SRI International Artificial Intelligence Center Technical Report*, 1997.
11. D. Oberle, S. Lamparter, S. Grimm, and D. Vrande. *Towards Ontologies for Formalizing Modularization and Communication in Large Software Systems*. Springer, 2006.
12. H. Paulheim and F. Probst. A Formal Ontology on User Interfaces Yet Another User Interface Description Language ? In *2nd Workshop on Semantic Models for Adaptive Interactive Systems (SEMAIS)*, 2011.
13. A. S. Rath. UICO: An ontology-based user interaction context model for Automatic Task Detection on the Computer Desktop. *CIAO '09: Proceedings of the 1st Workshop on Context, Information and Ontologies*, pages 1–10, 2009.
14. L. Sauermann, L. Van Elst, and A. Dengel. Pimo-a framework for representing personal information models. *Proceedings of I-Semantics*, 7:270–277, 2007.
15. W. M. P. van der Aalst and K. van Hee. *Workflow Management. Models, Methods, and Systems*. MIT Press Cambridge, 2002.
16. M. Wolpers, J. Najjar, K. Verbert, and E. Duval. Tracking actual usage: the attention metadata approach. 10:106, 2007.