The NExT Process Workbench: Towards the Support of Dynamic Semantic Web Processes

Abraham Bernstein, Michael Daenzer
University of Zurich
Institute for Informatics
8050 Zurich, Switzerland
{bernstein, daenzer}@ifi.unizh.ch

Abstract

Traditional process support systems offer the promise of software assembled from service elements. The typical approach is a static composition of atomic processes to more powerful services. In the real world, however, processes change over time: business needs are rapidly evolving and, thus, changing the work itself and relevant information may be unknown until workflow execution run-time. Hence, the traditional, static approach does not sufficiently address the need for dynamism. Based on applications in the life science domain this paper puts forward five requirements for dynamic process support system. Specifically, these demand a focus on a tight user interaction in the process discovery, composition, and execution phases. The system and the user establish a continuous feedback loop resulting in a mixed-initiative kind approach. We also present a prototype implementation NExT, which embodies this approach and present a preliminary validation based on a real-world scenario as well as a comparison with other process support tools.

1. An Illustrating Scenario - As is

Peter, our example scientist, wants to determine the 3D structure of a bio-molecule using NMR spectroscopy – a procedure that uses the magnetic resonances of protons within molecules to deduce the molecule's structure. First, he writes down the most complex parts of the NMR experiment's process sequence on a piece of paper. He skips all the standard steps, as he is able to perform them blindfolded. After having made his rough experimental plan, he feels confident to start the experiment. He inserts his sample into the spectrometer and runs the measurement – essentially the observation of proton resonances of the molecule triggered by a sequence of appropriate magnetic pulses. He then starts the analysis of the results: the measured resonance spectrum needs to be scaled and some properties must be retrieved by observation, all standard tasks, which Peter would really like to automate. Next he needs to use software tool A to count the peaks in his spectrum, which is followed by the identification of the resonances of the individual protons. Unfortunately, tool A is not able to do so. So Peter would like to invoke software tool B, which does, however, use a different input format. This forces him to transform the data manually before he can feed it to tool B and continue with his plan.

At some point, however he stumbles on a problem with his experiment, which he doesn't know how to solve. He vaguely remembers two publications about a similar problem and reads them. Unfortunately, he cannot find any information about the processes used. So, he decides to talk to his advisor, who is, however, visiting a conference overseas. As Peter does not have his experimental process plan in an electronic format, he needs to describe his intentions in prose, which leads to several misunderstandings during the phone call. Having overcome his experimental problem he studies some intermediate results returned from the spectrometer. Peter immediately realises that he forgot to repeat a proceeding measurement with adapted parameter values rendering the current measurement useless. So, he has to re-initiate the procedure with the new settings.

At the the end, Peter can complete his experiment with success and wants to publish his results. Ideally, he would like to publish the experimental procedure along with the actual results. His paper and pencil approach to experimental process control did not record all parameter settings. So, he skips that part and publishes his paper without any information about his pathway to the result.

2. Introduction

As Peter's example shows, conducting meaningful experiments (in the life science domain) as well as any ex-

ploratory activities (such as in complex insurance negotiations) requires a user to carry out a complex, long, and interrelated sequence of atomic tasks. During all stages in the life cycle of such tasks (see Figure 1), the actor is challenged by different issues that harden his work. During

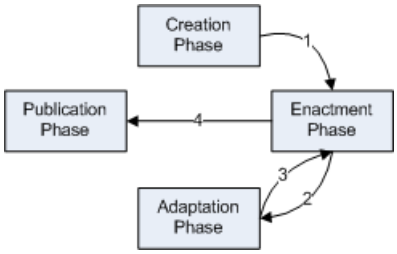

Figure 1. Life cycle of exploratory processes

the *creation phase*, the *process composition* of atomic tasks must be initially defined. In a first step abstract, atomic tasks are sequenced together conforming to the control and data flow specified by the user forming a higher-level composite service (*process choreography* [16]). The size of process compositions can be very large leading to complex interrelations making it difficult to gain and maintain an overview. Furthermore, processes (and their elements) may change their degree of specificity (see Figure 2) over time [3]: Underspecified processes can become well specified when more information becomes available and well-specified processes can become less specified over time (e.g., due to the occurrence of an exception). In Peter's case the experiment gains specificity when he adds some parameter settings; it looses specificity when he stumbles upon the problem that he solves with his advisor's help – thus, the process moves along the specificity frontier [3]. A system acting in domains whose processes show a varying degrees of specificity, must be able to handle processes in the whole spectrum as well as support their movement along the frontier. This issue is crucial for the correct definition of the interactions

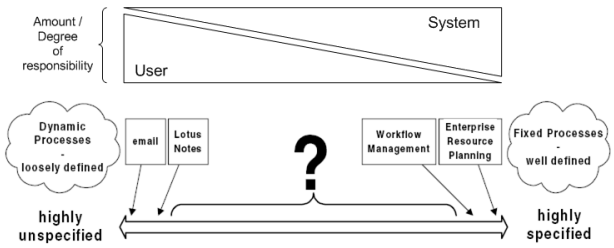

Figure 2. The Specificity Frontier [3]

In the *enactment phase*, the process is executed. Most executions of atomic processes can be automated or at least semi-automated using appropriate software tools or accessing/controlling some piece of hardware equipment. Usually, more than one potential realization (or implementation/grounding) does exist for one atomic process, so choosing the appropriate one (*process orchestration* [16]) can be complicated.

A process enters the *adaption phase* whenever an exception occurs during its execution. Exceptions may be raised, e. g., due to hardware malfunction, software crashes, or user interventions (see [8] for a full treatment of exception handling). When an exception is raised the execution is, usually, interrupted, the process must be adapted to handle the cause of the exception, and then its execution is resumed at the correct location. In particular the choice of the correct resumption point is extremely important in the case of long-running processes (e.g., a NMR measurement may run over several weeks) as any loss of intermediate results might cost a lot of time.

Once the experiment has finished, all its related data must be somehow documented (e.g., for publication in academia, to fulfill ISO 9000/9001 standards in industrial settings, and/or to record that due process was maintained in legal environments). We call this the *publication phase*. Traditionally, academic publications only include an informal prose report about the experimental setup but no concise formal description of the processes is published seriously hampering the possibility of replicating the experiments (one of the pillars of science).

Most current process support systems lack support for tight interaction between the user and the system during all the phases of the process's life-cycle. In our opinion they are, therefore, not flexible enough to optimally support users in highly dynamic domains, such as the the life sciences. We propose a novel system that focuses on the user as the driving factor and keeps him/her engaged in a tight interaction. Based on the system's domain knowledge and the explicitly given information, the system should provide contextual guidance to the user in all situations.

More specifically, based on the preliminary work [3], we proclaim that such a system has to fulfill the following requirements:

1. Support users throughout the process choreography and orchestration steps – Requirement $\mathcal{R}1$.

2. Support partial executions. An execution must be interruptible and resumable to make adaptations at runtime – Requirement $\mathcal{R}2$.

3. Integrate various deductive planners and reasoners to provide useful alternatives for the user – Requirement $\mathcal{R}3$.

4. Incorporate a *Case Base*. Then a *Case Based Reasoner [24]* can infer useful information from past cases (from

both best and worst practices) – Requirement $\mathcal{R}4$.

5. Support (semi-)automated *data mediation* to connect processes with different data formats which are transformable into each other – Requirement $\mathcal{R}5$.

The remainder of this paper is structured as follows: In Section 3 we operationalize the requirements into concrete foundational challenges for NExT, our prototype *N*ext-generation *Ex*periment *T*oolbox and is aimed at supporting execution of long running life science experiments. Section 4 then introduces the most important architectural and implementation aspects of NExT. A preliminary validation of the prototype in the context of the introductory scenario is discussed in Section 5, which is followed by a comparison with related work in section 6. We conclude with a summary and an outlook on future work.

3. Overall Operationalization of NExT

The requirements presented in the last section provide the basic requirements for building a flexible process support system. In order to assure a clear separation of concerns as well as domain independence in our NExT system we decided to divide it into two parts: the underlying knowledge bases (KBs) containing all the domain knowledge and a generic execution support system.

3.1. The Underlying Knowledge Bases

To fulfill the requirements outlined in the last section we will need three types of online KBs:

- A *Process Library* containing models for all processes (atomic processes including mappings to concrete realizations and templates for composite processes)
- A *Data Entity Library* containing models for all data entities
- A *Case Base* containing a collection of completed process executions with annotations (e.g., success/failure)

To address the two phases (choreography and orchestration) of $\mathcal{R}1$ the process library provides models for all atomic abstract tasks with a loose coupling to their concrete realizations (i.e., actual implementations) allowing for their dynamic reassignment. These can be categorized into different types depending on their access/call approaches and data input/output facilities. Facilities to model the mappings from the processes to those different types (such as web services or local applications) of realizations should be provided to ease the process model development.

Both the process and the data entity library provide all the data in a formal, machine readable language for planing and reasoning on currently running tasks addressing $\mathcal{R}3$. The data entity library contains information about data/object types to enable (semi-)automated data mediation in fulfillment for $\mathcal{R}5$. And, obviously, the case base enables both automated as well as human case based reasoning ($\mathcal{R}4$).

Note that due to the KBs NExT exhibits significant network effects in the micro-economic sense: the more people use the more attractive it becomes. If a sufficiently large group of people in any given domain can be convinced to publish their processes into these repositories then the possibility of increased knowledge exchange at the process level, the simplified possibility of collaboration in designing/executing processes, and their use as case bases and domain KBs for planners/reasoners would make the tool even more useful than in a single user setting. The more people would share their knowledge the more attractive its use.

3.2. The Execution System

The NExT execution system helps users during the whole process life cycle, from creation to publication. It guides the user by providing her with suggestions and ideas whenever she has to make decisions or explicitly requests help. This simplifies her task especially in the following phases:

- During the *process choreography* the system's degree of assistance ranges between suggestions, which processes are suitable for the next step, and the generation of whole process plans at once ($\mathcal{R}1$).
- During *process composition*. When two processes are chained together by a data flow and the types of their parameters are not identical (or "castable") then the system tries to resolve the mismatch or suggests solutions (e.g., by mediation) to the user ($\mathcal{R}5$).
- During the *process orchestration* the system will (1) guide the users to concrete realizations (or implementations) and (2) help them to decide which one is suitable under the given constraints and the user preferences ($\mathcal{R}1$)
- Whilst *handling exceptions* the system will help the user to adapt processes at runtime and ensure that the consistency is preserved ($\mathcal{R}2$).

In general, the user and the NExT system work hand-in-hand informing each other with newly discovered facts; an approach known as Mixed-Initiative planning and execution [24]. The more information and constraints the system receives from the user, the more knowledge it can infer and present to the user, which then can use this additional information to either retrieve even more information or make decisions, both of which become new input for the system.

User and system are, thus engaged in a continuous feedback loop. In addition, the system continuously monitors newly arriving information (such as detected exceptions) and initiates an interaction with the user where necessary.

The NExT user interface (UI) attempts to integrate all the necessary tools in one interface (following the workbench metaphor) acting as the control center for the user's daily work. As all UI features should be usable by non computer scientists (but domain experts) we tried to provide as simple as possible interaction approaches. A graphical data-flow style editor allows the user to easily create, start, pause, and adapt workflows. During executions it also provides her with feedback about the current state of the process and other process related information. Interactive browsers allow querying and browsing the KBs at any point in time and reasoners/planers/mediators act as wizard-like pop-ups to impart advise whenever asked.

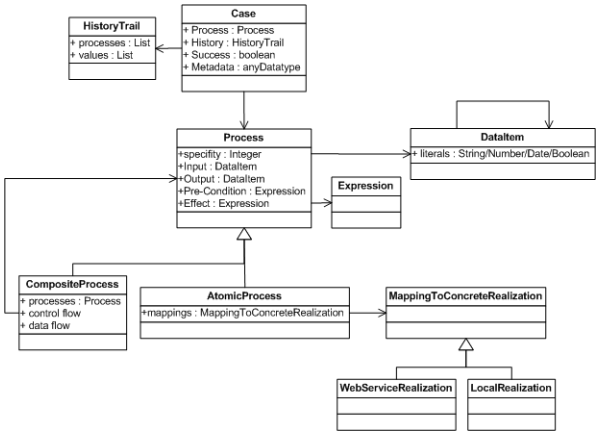

Figure 3. Simplified meta-model as an UML class diagram

4. The NExT Prototype Implementation

While the last section provided some additional detail about one possible operationalization of $\mathcal{R}1$-5 the provided information isn't sufficient to explain the actual functionality of our NExT system. This section, therefore, provides more background on its Meta-Model/Model and its system implementation.

4.1. The NExT Meta-Model

In order to ensure domain independence our system's process meta-model defines the system's view on both processes and data entities ($\mathcal{R}3$, $\mathcal{R}5$). Code was written in terms of meta-model concepts whereas applications may inherit from or extend the meta-model for their own purposes. Both meta-models and models must be encoded in a declarative, formal and machine readable family of languages that share the same conceptual ideas to ensure a seamless integration of the two models.

Figure 3 shows a simplified version of our meta-model. The concept *Process* generalises the two different types of processes: *AtomicProcess* and *CompositeProcess*. Inputs, outputs, pre-conditions, and effects (or post-conditions) define the semantics of a process and are the minimal properties to use AI planners [6] ($\mathcal{R}3$). The property *specificity* expresses the degree of specificity [3] for easy use in cost or weighting functions ($\mathcal{R}1$-3). An *AtomicProcess* additionally can be related with one or several mappings to concrete realizations ($\mathcal{R}1$). As an example, consider the abstract task (i.e., the *AtomicProcess*) "ResizePicture". Concrete realizations could be scripts using an image processor, a tool developed by oneself, or a web service providing this functionality. The mapping contains the specific how (and where) to invoke a realization. A *CompositeProcess* consists of a number of processes (potentially both atomic and composite), which are sequenced according to their specification in the process modeling language. For NExT we chose OWL-S [12] for that task, as most of the concepts we need are already integrated (see below for NExT's extensions to OWL-S). The *HistoryTrail* concept holds the execution history of a process. All atomic processes in their execution sequence and all intermediate values of almost[1] all parameters are stored together with the process description itself and other metadata forming a *Case* that is an element of the the Case Base ($\mathcal{R}4$). *DataItems* can be nested to compose complex types ($\mathcal{R}5$).

4.2. The NExT System Implementation

When implementing NExT we used a multi-tier architecture with currently four layers: the *Knowledge Bases* in the back end, a *Data Access* layer with the *Meta-Model/Model API*, a tier containing the system functionality (*Core*), and the *UI* tier (see also Figure 4).

The back end consists of the *Knowledge Bases* that are stored in a triple store (in our case Jena[2]). The *Data Access* layer enables the retrieval and publication of KB items (processes, data entities, and cases).

The *Meta-Model/Model API* is responsible for reading and writing the model data and (un-)marshalling them into/from normal Java objects. Most importantly, it includes a simple process execution engine that provides the execution primitives for the Execution Engine in the Core. To that

[1] Note that we allow certain information to be exempt from this storage requirement when explicitly requested by the user. This is sometimes necessary in scientific experiments when the amount of information becomes prohibitively large.

[2] see http://jena.sourceforge.net/

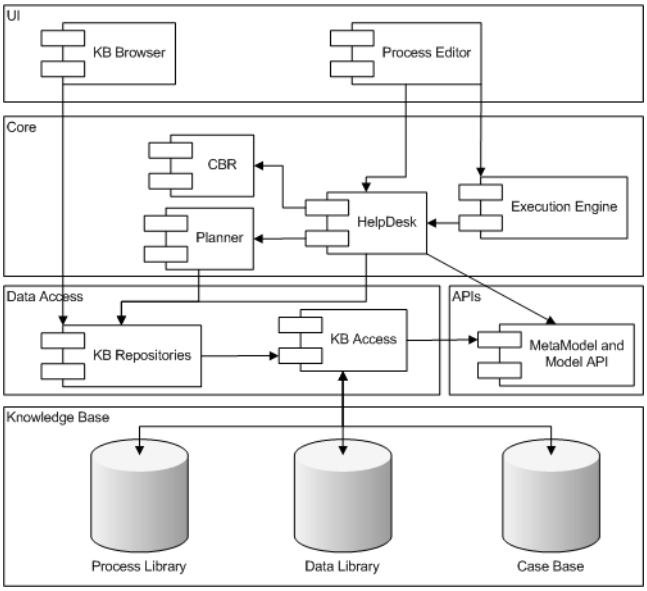

Figure 4. General architecture of NExT

end we extended the Mindswap OWL-S API[3] with the following two features: First, we added a new type of grounding that an atomic process directly to a Java method. Second, we augmented the API with a facility to interrupt and resume a process execution.

The *Core tier* consists of several components acting more or less directly on the model data. It could be used as a framework for any other process related system. The *HelpDesk* component coordinates all the guidance facilities, handles execution exceptions, manages the user interaction of all system components, and implements all the listeners observing new user inputs. To retrieve content for the user assistance, several types of inferring mechanism may be used. First, integrated deductive reasoners acting directly on the semantic model items. Specifically we used the Pellet reasoner [20] that came with the Mindswap API. Second, a Case Based Reasoner (the *CBR* component) can find past processes that are similar to the one in use. The current implementation of the CBR relies on SimPack[4] [4] to retrieve similar entities. The *planning* component provides a plug-in interface with which several AI planner can be integrated into the system. Their results will be merged together and then be presented to the user. Most of the standard AI planners return exactly one single solution – the "optimal" one – and assume that they have complete information about the world during planning time. Both properties do not hold in our application domains. First, users like to have several, ideally weighted, alternatives instead of one single solution. Second, if exceptions are thrown during the execution or processes are long-running, the state of the world may have changed in between. Last but not least, some information may not become available until runtime (as described by the specificity frontier [3] concept). A variety of existing planning tools have been already extended to operate on OWL-S or Semantic Web Services [22, 9, 19, 13, 17] and quite some research has been done to adapt planning to the changing information issue [10, 1]. We leverage that work by employing several of the above mentioned planers through our planner plug-in interface addressing $\mathcal{R}1, 3\text{-}5$.

The *Execution Engine* relies on the process execution services supplied by the Meta-Model/Model-API to enact (or execute) a workflow. Parallel executions are supported and the engine is able to interrupt a single execution for dynamic adaptation (i.e., when either an exception occurs or the user manually interrupts the execution). Continuation of a process is performed at the last known position where the process is still consistent (depending on how much its description was changed). Exceptions are tracked and feedback about the current state may be given to the user at all time for execution monitoring. Unfortunately, OWL-S doesn't specify exceptions. We, therefore, intend to adopt the exception conceptualisation of SWSF (Semantic Web Services Framework) [2] for OWL-S. The Execution Engine, thus, fulfills $\mathcal{R}2$.

The *UI tier* is based on the Eclipse[5] framework. It is built as a workbench integrating graphical tools for all important purposes. A process editor allows the graphical creation and editing of workflows as well as their initiation and interruption. The editor furthermore allows monitoring all process-related information such as partial results during execution. As mentioned above, the UI also provides suitable browsing/querying interfaces for the KBs.

5. Preliminary Validation – The Introductory Scenario Revisited

Peter is very happy, he just received a copy of the NExT Process Workbench. He has to perform exactly the same experiment again: the determination of the 3D view of a bio-molecule with a NMR spectrometer. He opens the process editor and the Knowledge Base Explorer and searches for similar projects in the past. He finds one and adopts the process sequence from this case (see Figures 5 and 6 for some example NMR process templates/cases contained in the NExT KBs). He can easily adapt the template to his needs and is warned by the system whenever he is violating the consistency of the plan. After finishing this task, he feels confident to start the experiments, so he puts his sample into the spectrometer and starts the process execution. Given the information in the process template the spectrometer is set-

[3] see http://www.mindswap.org/2004/owl-s/api

[4] see http://www.ifi.unizh.ch/ddis/research/semweb/simpack/

[5] see http://www.eclipse.org

up automatically and the measurement is started. As soon as the measurement has completed, the analysis of the results is initiated The spectrum is scaled and most of the desired property values are assigned by the system. Manual input is required only for a few tasks. Eventually, software tool A is automatically called to do some peak counting, the output data is transformed to the needs of tool B and passed to it to identify the resonances of the individual protons.

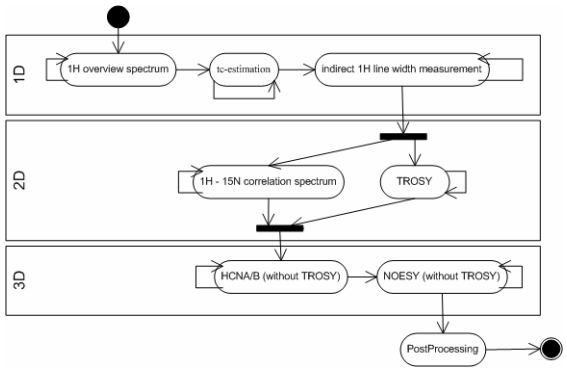

Figure 5. Overall experimental sequence for a NMR case

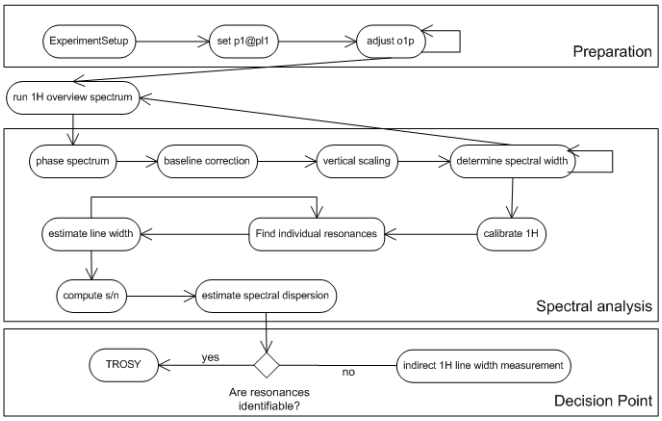

Figure 6. The control flow for a (simple) "1H overview" experiment (first step in Figure 5)

Peter continues the execution, but at some point he does not know how to proceed. He has no experience which measurements is most appropriate for this situation. He remembers two publications quite related to his research project, so he searches and finds them in Knowledge Bases. He then adopts the missing steps from their process sequences. In addition, he asks his advisor to verify the solution. As the advisor is at a conference overseas she connects to the shared work repository and can see exactly the same information on her own screen. She agrees on Peters solution, who directly continues with the execution. But the system immediately warns him that some data requirements for one of the inserted processes are not fulfilled. As a resolution the system suggests Peter to repeat a preceding measurement with adapted parameter values.

In the end, Peter can complete his experiment with success and needs to publish his results. He writes his prose publication as usual but additionally he uploads the whole case including all intermediate results, the trail of executed processes, and all additional information he entered into the system into a shared knowledge base of the journal. Evidentally he links the two things together. Furthermore, Peter is able to generalize a part of the process sequence for a certain type of bio molecules into a template and publishes it in a NMR-community maintained Knowledge Base.

6. Related Work

Various types of Process Support Systems have been developed in the past 30 years. Usually, however, they either support fixed, pre-defined, standard processes (e.g., workflow management Systems) or are specialized on ad-hoc dynamic processes providing no formalisms at all (such as e-mail or groupware). The former assist users during the workflow creation (processes are formally defined and, therefore, deductive reasoning upon them is possible), but usually do not allow run-time adaptations of the processes. Most traditional workflow support systems share the focus on control flow as shown for example in [23]. The latter ones are based on very informal, ad-hoc processes. Reasoning facilities are "out of focus" for such systems (as they lack formal process specifications), but their ability of process adaptations at run-time is their key feature.

More modern systems from the scientific community are oriented towards data-flow, service orientation, and grid computing. Prominent representatives thereof are Kepler [11], Pegasus [7], and Taverna [15] . All of them provide all the basic functionality. All these tools offer nice graphical editors to create processes, help during the discovery of appropriate atomic tasks by some sort of matchmaking algorithms and are able to execute a process. But, none of them is focused on the tight user integration and therefore most of our requirements are not fulfilled (see Table 1 for a comparison of all systems in the light of the requirements). Taverna and Kepler enable users to interrupt and resume process executions. In Taverna the user is able to edit data during an interruption in a debugger-like interface, but in both tools adaptations to control or data flow are not possible. Pegasus is the only of the examined modern system, which makes the difference between an abstract process and its realization. Pegasus uses a partial-order planning [6] algorithm to help users compose processes in an automated way and in combination with Virtual Data System (VDS) [25] some

interfaces support for data mediation are provided.

Name	$\mathcal{R}1$	$\mathcal{R}2$	$\mathcal{R}3$	$\mathcal{R}4$	$\mathcal{R}5$
Taverna	+	partially	-	-	+
Kepler	+	partially	-	-	-
Pegasus	+	-	+	-	+
Web Service Composer	+	-	+	-	-
WSMX	+	-	-	-	+

Table 1. Feature matrix for related work

The Web Service Composer [21] is a prototype which aims at guiding the user iteratively in the process composition. The user starts with a first process step and the tool filters finds all processes that can be chained with the initial one based on their fit in terms of inputs and outputs. It is to ensured that pre- and post-condition are structurally satisfied, i.e., all variables used in expressions must be known at that time. The process compositions can be executed but any further assistance or dynamic adaptations at run-time are out of scope of the tool.

WSMX[6] is the reference implementation for WSMO and is conceptually comparable with the Meta-Model/Model API of our system. It does not directly provide end-user tools (although some WSMO construction tools have been built) but a virtual machine to run WSMO processes. An important built-in functionality are so-called data mediators, which find a mapping between two data entities (both must be modeled in an ontology) at design-time and then actually transforms the entity at run-time. We did not find any information about WSMO's support for partial executions.

7. Future Work/Conclusion

In future, we intend to further develop our NExT prototype implementation. We are also planning to focus in a second step on adding further planning and reasoning technologies. Furthermore, UI Integration for the interfaces of the referenced tools in the concrete realizations (especially for semi-automated tasks) and intuitive means for user interaction are interesting, open research areas and a key factor for success of any process support system designed for users, which are not workflow experts. For example, how should a non-expert enter expressions in KIF [5], SPARQL [18] or any other logic based language without knowing syntax and semantics of these languages by heart? Additional open issues are the integration of exception mechanisms and enhanced data mediation facilities into the conceptual framework, inspired by the other approaches for semantic process frameworks such as SWSF [2] and WSMO [14]. Last, and most importantly, we intend to deploy NExT in a life science environment, and observe its practical usage for complex experimental analysis.

In this paper, we presented an approach to resolve the non-trivial tasks of process composition, execution, runtime adaptation, and publication in complex domains such as the life sciences. Human beings like to be in control of the creative and complex parts of their work. We, therefore, believe that a process support system should have a tight interaction with its human users. They should be assisted throughout the whole process life cycle and standard tasks should be executed automatically. But they need the ability to override all decisions and take over the control. Hence, a permanent feedback loop must be established to overcome the complexities of the specificity frontier and react adequately at all times on process specificity changes.

As our main contributions we have developed five requirements for process support systems in complex experimental domains. We have, furthermore, shown a basic architecture and key implementation elements of our NExT process support system based on Semantic Web technologies and AI planning and reasoning methodologies (planners, Case-Based Reasoning) that implements our vision.

[6] See http://www.wsmx.org for detailed information

8. Acknowledgements

We would like to thank Konstantin Pervushin for sharing his knowledge about NMR spectrography with us and for his fruitful ideas. Especially the illustrating scenario could not have been drawn without his help.

References

[1] T.-C. Au, U. Kuter, and D. Nau. Web service composition with volatile information. In *Proceedings of the International Semantic Web Conference (ISWC)*, 2005.

[2] S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIlraith, D. McGuiness, J. Su, and S. Tabet. Semantic web services ontology. Technical report, 2005.

[3] A. Bernstein. How can cooperative work tools support dynamic group processes? bridging the specificity frontier. In *Proceedings Computer Supported Cooperative Work (CSCW 2000)*. ACM Press, 2000.

[4] A. Bernstein, E. Kaufmann, C. Kiefer, and C. Bürki. Simpack: A generic java library for similiarity measures in ontologies. Technical report, Department of Informatics, University of Zurich, 2005.

[5] M. R. Genesereth. Knowledge interchange format. Technical report, Draft proposed American National Standard (dpans), 1998.

[6] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning, theory and practice*. Elsevier, 2004.

[7] Y. Gil, V. Ratnakar, E. Deelman, M. Spraragen, and J. Kim. Wings for pegasus: A semantic approach to creating very large scientific workflows. In *Proceedings of the OWL: Experiences and Directions 2006 (OWL-06), Athens, GA*, 2006.

[8] M. Klein and C. Dellarocas. A knowledge-based approach to handling exceptions in workflow systems. *Computer Supported Cooperative Work: The Journal of Collaborative Computing*, 9(3-4):399–412, 2000.

[9] M. Klusch, A. Gerber, and M. Schmidt. Semantic web service composition planning with owls-xplan. In *1st International AAAI Fall Sympsoium on Agents and the Semantic Web*, 2005.

[10] U. Kuter, E. Sirin, B. Parsia, D. Nau, and J. Hendler. Information gathering during planning for web service composition. *Journal of Web Semantics*, 3(2), 2005.

[11] B. Ludscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system. *Journal for Concurrency and Computation: Practice Experience, Special Issue on Scientific Workflows*, 2005.

[12] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIllraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. Owl-s semantic: Markup for web services. 2004.

[13] S. McIlraith and T. Son. Adapting golog for composition of semantic web services. In *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002)*, 2002.

[14] A. Mocan and E. Cimpian. Wsmx data mediation. Technical report, ESSI WSMO working group, 2005.

[15] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: A tool for the composition and enactment of bioinformatics workflows bioinformatics journal 20(17) pp 3045-3054, 2004. *Bioinformatics Journal*, 20(17):3045–3054, 2004.

[16] C. Peltz. Web services orchestration and choreography. *Computer, Innovative Technology for Computing Professionals*, 2003.

[17] S. R. Ponnekanti and A. Fox. Sword: A developer toolkit for web service composition. In *Proceedings International WWW Conference(11), Honolulu, Hawaii, USA*, 2002.

[18] E. Prud'hommeaux and A. Seaborne. Sparql query language for rdf. Technical report, W3C Candidate Recommendation, 2006.

[19] M. Sheshagiri, M. desJardins, and T. Finin. A planner for composing service described in daml-s, workshop on planning for web services. In *International Conference on Automated Planning and Scheduling*, 2003.

[20] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: a practical owl-dl reasoner. *Journal of Web Semantics*.

[21] E. Sirin, B. Parsia, and J. Hendler. Composition-driven filtering and selection of semantic web services. In *AAAI Spring Symposium on Semantic Web Services*, 2004.

[22] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. Htn planning for web service composition using shop2. *Journal of Web Semantics*, 1(4):377–396, 2004.

[23] W. van der Aalst. Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems. Web Services - Been there done that? Trends Controversies*, 2003.

[24] M. M. Veloso, A. M. Mulvehill, and M. T. Cox. Rationale supported mixed-initiative case-based planning. In *IAAI-97, Innovative Applications of Artificial Intelligence*, 1997.

[25] Y. Zhao, M. Wilde, I. Foster, J. Voeckler, J. Dobson, E. Gilbert, T. Jordan, and E. Quigg. Virtual data grid middleware services for data-intensive science. In *Middleware 2004, Concurrency, Practice and Experience*, 2004.