The NExT System: Towards True Dynamic Adaptations of Semantic Web Service Compositions (System Description)

Abraham Bernstein and Michael Daenzer

University of Zurich, Department of Informatics, 8050 Zurich, Switzerland
{bernstein,daenzer}@ifi.unizh.ch

Abstract. Traditional process support systems typically offer a static composition of atomic tasks to more powerful services. In the real world, however, processes change over time: business needs are rapidly evolving thus changing the work itself and relevant information may be unknown until workflow execution run-time. Hence, the 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 systems*. These demand a focus on a tight user interaction in the whole process life cycle. The system and the user establish a continuous feedback loop resulting in a mixed-initiative approach requiring a *partial execution and resumption feature* to adapt a running process to changing needs. Here we present our *prototype implementation* NExT and discuss a preliminary validation based on a real-world scenario.

1 An Illustrating Scenario - As is

Peter, the chemist in our scenario, needs to determine the 3D structure of a bio-molecule using NMR spectroscopy. Without having IT support, he uses his paper lab book to construct a rough experimental plan. He then starts the experiment. Only, he forgets to calibrate the spectrometer, a fact he quickly realises as the spectrometer returns first data, which shows a systematic and continuous shift over all values. At some later point, Peter stumbles on a problem with his experiment, which he does not know how to solve. He is unable to interpret a spectrum correctly and therefore to choose which measurement to perform as the next step. He reads a publication about a similar problem and makes a lengthy (formal descriptions are hard to explain in prose form) telephone call with his advisor, which is visiting a conference overseas. Anyhow, his advisor can help him and following his lead, he studies some intermediate results returned from the spectrometer. Peter then realises that he forgot to repeat a proceeding measurement with adapted parameter values rendering the current measurement totally useless. But even worse, he did not store the intermediate results during the execution, so he has to restart the execution from scratch.

2 Introduction

The scenario shows that conducting meaningful experiments or exploratory activities requires a user to construct a complex and long-running sequence of atomic tasks which may have to be changed during all phases in their life cycle (*process choreography [1]).* Their size can be very large leading to long and complex interrelations. Furthermore, processes and their elements may change their degree of specificity (see Figure 1) over time: Underspecified processes can become well specified when more information becomes available and well-specified processes can become less specified (e.g., due to exceptions) – thus, the process moves along the Specificity Frontier [2]. A system acting in domains whose processes show varying degrees of specificity and dynamically move along the frontier must conform to such behaviour.

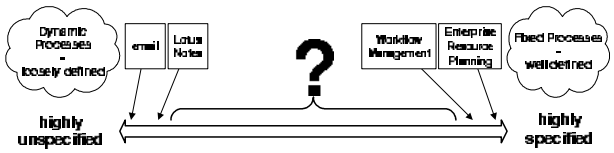

Fig. 1. The Specificity Frontier [2]

Usually, several potential realizations exist for an atomic task (such as Web or Grid Services or local procedure calls), so choosing the appropriate one (*process orchestration* [1]) turns out to be non-trivial. At run-time, exceptions can be thrown (e. g., hardwre malfunctions, software crashes), unforeseeable events may take place or the user wants to *intervene* when he observes something unusual, forcing the execution to halt and the system to react accordingly. The process must then be adapted in some way preserving its correctness and consistency. Finally, the execution must be resumed at the correct and optimal resumption point. Once the experiment has finished, all its related data must be documented (e.g., for publication in academia or to record that the process was maintained in legal environments).

In our opinion, a process support system acting in highly dynamic domains must focus on the user and keep 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, especially in the complex creative phases of his work. More specifically, based on the preliminary work [2], we proclaim that such a system has to fulfill the following requirements:

$\mathcal{R}1$: Support users throughout the process choreography and orchestration steps.
$\mathcal{R}2$: Support partial executions and dynamic adaptations at run-time.
$\mathcal{R}3$: Integrate reasoners and planners to provide useful alternatives for the user.
$\mathcal{R}4$: Incorporate a *Case Base*. Then a *Case Based Reasoner [3]* can infer useful information from past cases (from both best and worst practices).

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

In this paper we will present an overall approach for a process support system addressing these requirements. We focus on the second one and will show in more detail how partial executions, run-time adaptations and changes to parameter values can be supported. The remainder of this paper is structured as follows: In Section 3 we operationalize the requirements into concrete foundational challenges for our prototype NExT (*N*ext-generation *Ex*periment *T*oolbox). 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, 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

In order to assure and a clear separation of concerns, we divided NExT into two parts: the underlying knowledge bases (KBs) containing all the domain knowledge and a generic execution support system. The content of these KBs is provided in a formal, machine readable language, which is a pre-requisite for planning and reasoning ($\mathcal{R}3$, $\mathcal{R}4$). We identified three types of entities to store in separate online KBs: First a *Process Library* with models for all atomic tasks and templates for composite processes with a loose coupling to their concrete realizations allowing for their dynamic reassignment ($\mathcal{R}1$). Second, a *Data Entity Library* containing models for all data/object types to enable (semi-)automated data mediation in fulfillment for $\mathcal{R}5$. Last, but not least, a *Case Base* containing a collection of completed process executions 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 it the more attractive it becomes. If a sufficiently large group of people in a given domain publish their processes into the repositories then the possibility for knowledge exchange increases, collaborating in designing/executing processes is simplified, and their use as case bases and domain KBs increases the quality and diversity of planner/reasoner results.

We follow an approach known as Mixed-Initiative planning and execution [3]; the user and the NExT system work hand-in-hand informing each other with newly discovered facts. The more information and constraints the system receives from the user, the more (implicit) knowledge it can infer and present to him. He 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 whenever necessary.

NExT guides the user by providing suggestions whenever she has to make decisions or she explicitly requests help. 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 the *process orchestration* the system will (1) guide the users to concrete realizations and (2) help them to decide which one is suitable under the given constraints and user preferences ($\mathcal{R}1$). When two processes are chained together by a data flow and the types of their parameters are not "castable", then the system tries to resolve the mismatch or suggests solutions to the user ($\mathcal{R}5$). The execution history is recorded and contains the execution sequence of atomic tasks, the links to used realizations, and all intermediate results. This is an apparent prerequisite to build up cases ($\mathcal{R}4$) for CBR.

The NExT user interface (UI) attempts to integrate all the necessary tools in one common interface (the workbench metaphor). NExT's target audience are not computer scientists (but domain experts), so 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 and shows also the current state of the process during execution. 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.

3.1 Supporting Partial Executions and Adaptations

In contrast to pure static workflows, dynamically evolving processes in most cases cannot be fully specified before the start of the execution (e.g., some relevant information becomes only available at run-time). Therefore, we allow the user to start executing such processes, at least as long as the first steps in the sequence are well specified. Over time the amount of information rises and the process specification can be improved iteratively. Nevertheless, every problem that leads to a failure at runtime must be resolved. Hence, our concept of partial executions consists of four elements: (1) errors in the process specifications and/or exceptions and events must be detected before they affect the execution, (2) the process execution must be interruptible, (3) the user must be able to adapt the process to solve the problem, and (4) the execution must be re-continuable at a correct and optimal point to ensure the overall process consistency.

The process specification is validated each time before the execution starts (or re-continues) and at run-time, exceptions and events are caught by an exception handler. For further handling, we developed an ontology of possible incidents

Exception	**Resolution Strategy**
Hole in the sequence	1. Call planner to provide alternatives to fill the gap 2. Ask user to define a realization manually
Missing parameter makes a condition unsatisfiable	1. Query KB for processes, that produce the missing variables 2. Relax the condition 3. Remove processes whose effects make the condition unsatisfiable 4. At run-time, instantiate an input and let the user enter its value.

Table 1. Excerpt of the problem ontology including the problem resolution strategies

combined with adequate (semi-)automated resolution strategies (see Table 1). After catching a problem, NExT exploits this ontology to map each problem to an incident and determines then the priority for the problem resolution. Whenever a severe incident is detected that endangers the immediate continuation of the process the respective resolution strategy is applied instantly. On the other hand, minor, not time-critical problems are simply reported to the user which then can trigger the resolution manually (or the incident's severity rises over time above a threshold and then needs to be resolved immediately).

In most cases atomic tasks will not be interruptible when already under execution (except they explicitly support this behaviour). In most cases this issue can be addressed by interrupting the execution of the overall process when the execution of the current atomic task finishes. If the cause for the interruption is related to the outcome of the atomic task's execution then its outcome will have to either ignored, undone, or taken into account when it finished (in fact, this is an instance of the Specificity Frontier). Consider this logistics scenario: The plane transporting a piece of cargo for us is already airborne and we hear that the cargo staff at the destination airport is on strike. Thus, it will not be delivered at the demanded time, which is a hard constraint from our costumer. Since, it is not in our power to reroute the plane we have to adapt our process to the new circumstances.

Whenever possible the strategies attempt an automated, systems-led resolution of the exception. If this fails or the user intervenes, she is integrated in the loop (usually when too little information is known for the incident's resolution). We found that the majority of the resolution strategies include changes in the parameter values or adaptations of the process's control and/or data flow. Thus NExT must support such change operations and guide the user by the same mechanisms as during process creation phase. In addition it must be ensured that the process's new execution plan is consistent and of its execution trail/history remains correct and consistent. Before re-continuation of the process, the correct and optimal resumption point must be found. Whenever processes or parameter values that already were executed respectively computed are changed, it must be computed wether the execution path is still correct. If not, some processes must be rolled back to start over at a previous stage of the execution.

4 The NExT Prototype Implementation

In order to ensure domain independence our system's process meta-model defines the system's view on both processes and data entities ($\mathcal{R}3$, $\mathcal{R}5$ - Planner, Mediation). Code was written in terms of meta-model concepts whereas applications may inherit from or extend the meta-model for their own purposes. We describe processes by their IOPE, meaning the (semantic) notion of inputs, outputs, preconditions and effects (or post-conditions) and encode them in a declarative, formal and machine readable language. These are the minimal properties to use AI planners [4] ($\mathcal{R}3$). We furthermore differentiate between an *AtomicTask* and a *CompositeProcess*, whereas only the former can be related with one or sev-

eral mappings to concrete realizations ($\mathcal{R}1$). The mapping contains the specific how (and where) to invoke a realization. A *CompositeProcess* on the other hand consists of a sequence of processes (potentially both atomic and composite). We have chosen to use OWL-S [5], because it supports most of the concepts we need out-of-the-box. When the execution of a process starts, the *HistoryTrail* is attached. All atomic tasks in their execution sequence and all intermediate values of all parameters are stored and define hereby a *Case* ($\mathcal{R}4$). *DataItems* can be nested to compose complex types ($\mathcal{R}5$ - mediation).

As our process execution engine we extended the Mindswap OWL-S API[1] with two features: First, we added a new type of grounding that an atomic task directly maps to a Java method. Second, we augmented the API with a facility to interrupt and resume a process execution. NExT, furthermore, provides a component to retrieve content for the user assistance ($\mathcal{R}1$, $\mathcal{R}3$-$\mathcal{R}5$) and a second component controlling the partial execution and dynamic adaptation aspect ($\mathcal{R}2$), which we present in more detail in the next section. The guidance component integrates several types of inferencing mechanisms:

- Deductive reasoners acting directly on the semantic model items. Specifically we used the Pellet reasoner [6] that came with the Mindswap API.
- A Case Based Reasoner can find past processes similar to the one in use. The current implementation relies on SimPack[2] [7] to retrieve similar entities.
- A plug-in interface to integrate several AI planners suitable for web service composition [8,9,10,11,12] into the system. Herby we can exploit their specialization on a certain planning aspect (e.g. to use planners addressing the changing information issue [13,14].

NExT is based on the Eclipse[3] 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, their initiation and interruption. as well as monitoring all process-related information such as partial results during execution.

4.1 Supporting Partial Executions and Adaptations

We implemented a hierarchy with specific handlers for each type of incident in our ontology. These handlers encapsulate the incident itself, its severity, and implement its resolution strategy. To ease the development of these strategies general facilities for common steps are provided by the NExT system (such as UI widgets for user interaction or encapsulations for standard interactions with planners). We then extended the Mindswap OWL-S API to perform consistency checks on OWL-S process descriptions for design-time detection of problems and improved the exception handling within the execution engine for run-time detection. Both methods return an instance of incident stubs (or a list thereof).

[1] see http://www.mindswap.org/2004/owl-s/api
[2] see http://www.ifi.unizh.ch/ddis/research/semweb/simpack/
[3] see http://www.eclipse.org

Depending on its severity, the incident is either added to a warning list for detached resolution or the resolution strategy is immediately applied. As long as the problems are not resolved, The execution is postponed (when not yet started) or stays interrupted as long as not all severe problems are resolved.

Once the execution (re-) starts, the correct and optimal resumption point must be computed. If all changes took place after the current execution point, then we can simply continue the process. Else the algorithm attempts to roll back all the effects of the computation by applying the following strategy to each process step backwards until the first change:

1. When an inverse process is specified, invoke it. Proceed with next step.
2. When the process triggered no changes in the world state besides IO transformations, the corresponding values are set back. Proceed with next step.
3. The user is asked to perform the roll-back manually. To suggest potential solutions, the process library is queried to find a process with reversed input/output and pre-/post-condition.
4. Abort the execution.

Note, that we must consider that massive amounts of data can be generated during the execution. Hence, storing all intermediate results on all atomic tasks is unpractical. We therefore let the user define storage points in the process sequence at which the intermediate results are written on disk. Second, note, that finding the correct termination point for the strategy above shows some complexity too. Parallel execution of steps or loop construct may introduce dependencies between steps, which must be taken into account. The actually implemented algorithm takes these two points into consideration. With all these considerations, we propagate to fulfill $\mathcal{R}2$.

5 Preliminary Validation – The Introductory Scenario Revisited

Let us have a second look at our scenario. Peter conducts the same experiment, this time with a copy of NExT. First, he finds a similar project in the past, adopts its process sequence (see Figure 2 for a screenshot of NExT) and adapts it slightly to his needs. The "calibrate spectrometer" process is part of a) the pre-condition of the "run measurement" process and b) the standard "setup spectrometer" process template, so this time Peter does not forget this step. All the steps that can be automated such as spectrometer calibration or some simple analysis steps are executed automatically, but still some tasks need to be performed manually. Peter though encounters the same problem as before. But this time, the system provides him several potential solutions and he chooses the correct alternative amongst them. NExT re-sets the execution pointer to the correct position and continues the experiment avoiding its restart from scratch. In the end, Peter completes his experiment with success and much faster than earlier. In addition to his prose report, he uploads the whole case including all intermediate results, the history trail, and all additional information into a

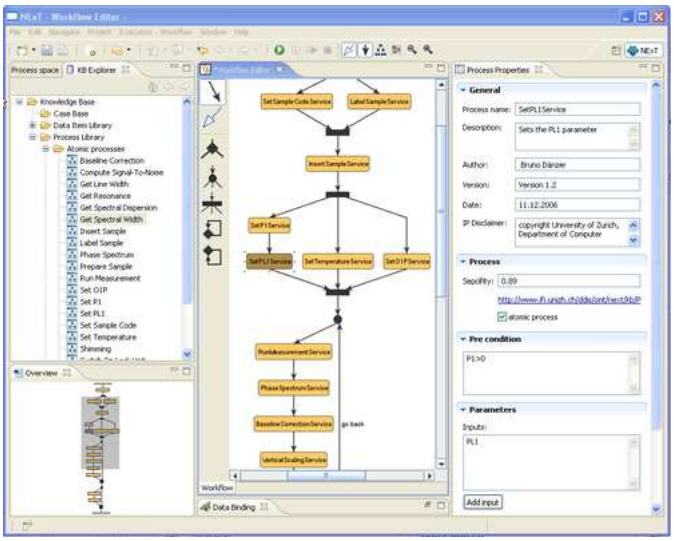

Fig. 2. Screenshot of NExT with the experimental sequence for a NMR case

shared knowledge base of the journal. 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

Most of the Process Support Systems that have been developed in the past 30 years support either fixed, pre-defined, standard processes (e.g., workflow management systems) or informal ad-hoc dynamic processes (such as e-mail or groupware). The former use formal process definitions and can thus assist users during the workflow creation whereas the latter are not bound to strict rules to ensure flexible process adaptations at run-time. Only a few systems provide the base for both. The FAR [15] system implements an exception handler based on Event-Condition-Action (ECA) rules defined in a specific exception specification language. ADEPT$_{flex}$ [16] is based upon a graph-based workflow model and includes a complete and minimal set of (dynamic) change operations such as task insertion or deletion. Consistency and correctness are preserved hereby.

Modern systems from the life science community are oriented towards service orientation and grid computing. Prominent representatives thereof are Kepler [17], Pegasus [18], and Taverna [19]. They all provide the basic functionality to help users in the process life cycle, but none of them is focused on highly dynamic processes and tight user integration. Both Taverna and Kepler allow the user to manually pause an execution, Taverna can re-assign intermediate results during an interruption and Kepler allows in addition adaptations to the control and data flow. Pegasus on the other hand differentiates between the process and

its realization, uses a partial-order planning [4] algorithm for guidance in the process composition and in combination with Virtual Data System [20] some interfaces support for data mediation are provided.

7 Future Work/Conclusion

In future, we want to deploy NExT in a life science environment to observe its practical usage for complex experiments. We plan to extend OWL-S by integrating the concepts of exceptions and events into the language. This would enable reasoning upon these concepts and thus improve the user guidance facilities. Furthermore, we will incrementally extend and refine our incident ontology and NExT's facilities for applying the resolution strategy. Also, we hope to exploit the ongoing research on both AI and non-AI composition algorithms to offer further guidance to users on the process composition and adaptation steps.

In this paper, we presented an approach for a process support system that assists its users throughout the whole process life cycle from creation to enactment, adaptation and publication in the end. The system aims at domains confronted with complex, long-running and highly dynamic processes. The process support system maintains a tight interaction with its human users: they want to be assisted in the creative work parts and they need to have the full control, but simple and monotonic tasks should be executed automatically to hold off the user from these time-consuming tasks.

As our main contribution we 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. We especially focused on the *partial execution feature* ($\mathcal{R}2$) and showed how we detect problems, exceptions, and events at run-time (as well in design-time), allow for appropriate adaptations in the process, and resume the execution at the correct and optimal resumption point. We hope that such systems will enable the practical use of Semantic Web Services system in practice.

Acknowledgments
We are deeply indebted to Professor Konstantin Pervushin for his expertise in the NMR spectroscopy domain and to Professor Josef Joller and his students Markus Krähenbühl, Georg Kunz and Franco Sebregondi for their contribution in the runtime adaptation component.

References

1. Peltz, C.: Web services orchestration and choreography. Computer, Innovative Technology for Computing Professionals (2003)
2. Bernstein, A.: How can cooperative work tools support dynamic group processes? bridging the specificity frontier. In: Proceedings Computer Supported Cooperative Work (CSCW 2000), ACM Press (2000)

3. Veloso, M., Mulvehill, A., Cox, M.: Rationale supported mixed-initiative case-based planning. In: IAAI-97, Innovative Applications of Artificial Intelligence. (1997)
4. Ghallab, M., Nau, D., Traverso, P.: Automated Planning, theory and practice. Elsevier (2004)
5. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIllraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: Owl-s: Semantic markup for web services. (2004)
6. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: a practical owl-dl reasoner. (Journal of Web Semantics)
7. Bernstein, A., Kaufmann, E., Kiefer, C., Bürki, C.: Simpack: A generic java library for similiarity measures in ontologies. Technical report, Department of Informatics, University of Zurich (2005)
8. Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: Htn planning for web service composition using shop2. Journal of Web Semantics **1**(4) (2004) 377–396
9. Klusch, M., Gerber, A., Schmidt, M.: Semantic web service composition planning with owls-xplan. In: 1st International AAAI Fall Sympsoium on Agents and the Semantic Web. (2005)
10. Sheshagiri, M., desJardins, M., Finin, T.: A planner for composing service described in daml-s. In: International Conference on Automated Planning and Scheduling. (2003)
11. McIlraith, S., Son, T.: Adapting golog for composition of semantic web services. In: Proceedings of the 8th Intl. Conference on Knowledge Representation and Reasoning. (2002)
12. Ponnekanti, S.R., Fox, A.: Sword: A developer toolkit for web service composition. In: Proceedings Intl. WWW Conference. (2002)
13. Kuter, U., Sirin, E., Parsia, B., Nau, D., Hendler, J.: Information gathering during planning for web service composition. Journal of Web Semantics **3**(2) (2005)
14. Au, T.C., Kuter, U., Nau, D.: Web service composition with volatile information. In: Proceedings of the International Semantic Web Conference (ISWC). (2005)
15. Casati, F., Ceri, S., Paraboschi, S., Pozzi, G.: Specification and implementation of exceptions in workflow mangament systems. ACM Transactions on Database Systems **24**(3) (1999) 405–451
16. Reichert, M., Dadam, P.: Adeptflex - supporting dynamic changes of workflows without losing control. Journal of Intelligent Information Systems - Special Issue on Workflow Managment **10**(2) (1998) 93–129
17. Ludscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger-Frank, E., Jones, M., Lee, E., Tao, J., Zhao, Y.: Scientific workflow management and the kepler system. Journal for Concurrency and Computation: Practice and Experience, Special Issue on Scientific Workflows (2005)
18. Gil, Y., Ratnakar, V., Deelman, E., Spraragen, M., Kim, J.: Wings for pegasus: A semantic approach to creating very large scientific workflows. In: Proceedings OWL: Experiences and Directions. (2006)
19. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A., Li, P.: Taverna: A tool for the composition and enactment of bioinformatics workflows bioinformatics journal 20(17) pp 3045-3054, 2004. Bioinformatics Journal **20**(17) (2004) 3045–3054
20. Zhao, Y., Wilde, M., Foster, I., Voeckler, J., Dobson, J., Glibert, E., Jordan, T., Quigg, E.: Virtual data grid middleware services for data-intensive science. In: Middleware 2004, Concurrency, Practice and Experience. (2004)