Semantic Web Services Challenge:
Proceedings of the 2008 Workshops

Edited by Charles Petrie

Abstract The Semantics Web Services Challenge is a unique initiative among such academic and industrial tests. It is not a contest with winners but rather a certification process that evaluates certain technologies against problems with both an objective evaluation and a consensus judgment. This report consists of the proceedings of the two workshops held in 2008.
Table of Contents

- “Introduction to the SWS Challenge 2008 Workshop Proceedings” - Charles Petrie

ESWC 2008 Papers

- “Debugging OWL Ontologies - A Reality Check”: Heiner Stuckenschmidt
- “Semantic System Integration: Incorporating Rule-based Semantic Bridges into BPEL Processes”: Nils Barnickel, Ralf Weinand, Matthias Fluegge
- “Semantic Web Service Discovery and Selection: a Test Bed Scenario”: Alessio Carenini, Dario Cerizza, Marco Comerio, Emanuele Della Valle, Flavio De Paoli, Andrea Maurino, Matteo Palmonari, Matteo Sassi, and Andrea Turati
- “Synthesizing the Mediator with jABC/ABC”: Tiziana Margaria, Marco Bakera, Harald Raffelt, and Bernhard Steffen
- “Abductive Synthesis of the Mediator Scenario with jABC and GEM”: Christian Kubczak, Tiziana Margaria, Matthias Kaiser, Jens Lemcke, and Björn Knuth

ISWC 2008 Papers

- “Model-driven Service Integration using the COSMO Framework”: Dick A.C. Quartel, Stanislav Pokraev, Teduh Dirghahayu, Rodrigo Mantovaneli Pessoa, and Marten van Sinderen
- “Advances in Solving the Mediator Scenario with jABC and jABC/GEM”: Jens Lemcke, Matthias Kaiser, Christian Kubczak, Tiziana Margaria, and Björn Knuth
Introduction to the SWS Challenge 2008 Workshop Proceedings

Charles Petrie, Stanford Logic Group

The Challenge is a unique initiative among such academic and industrial tests. It is not a contest with winners but rather a certification process that evaluates certain technologies against problems with both an objective evaluation and a consensus judgment. The problems come from both the organizers and the participants (whom are not completely represented by these authors as some valuable contributors to the 2008 workshops did not write papers.)

One of the drawbacks of conventional workshops is that papers are presented but apart from peer-review, there is no test of the claims in the papers. The Challenge offers an important improvement for the scientific process because the technologies can actually be tested in a consensus environment. And one fundamental result of the Challenge to date is that such claims are actually much harder to achieve in practice than anyone would suspect by just reading normal papers.


This technical report contains the actual papers of the two 2008 workshops. The papers are not numerous because the Challenge workshops consist largely of work: evaluating the previous work and conducting "surprise problems" to test the flexibility of the various technologies. And the participant group changes only slowly so much of the work proceeds incrementally.

The first workshop was held at the European Semantic Web Conference in Tenerife, Spain (ESWC 2008). The first paper of this workshop was unusual for the challenge in that made no claims for a technology, but rather was a survey and analysis. "Debugging OWL Ontologies - A Reality Check" by Stuckenschmidt, is a different kind of evaluation in which various supporting technologies are tested on common problems. The results are interesting: it develops that even well-founded systems produce wrong results.

"Semantic System Integration: Incorporating Rule-based Semantic Bridges into BPEL Processes" by Barnickel, Weinand, and Fluegge from FOKUS was a new technology participant in 2008. This is a model-based technology that solved the first mediation scenario problem.

"Semantic Web Service Discovery and Selection: a Test Bed Scenario" by Caremini et al. is a new proposal for scenario problem set. This is revision of a proposal made the previous year. Many of the main problems were remedied, but the discussion continued through the 2nd 2008 workshop.

"Synthesizing the Mediator with jABC/ABC" by Margaria et al., of the Dortmund/Potsdam group, described and tested a new use of the jABC technology. The basic jABC technology is based upon modeling,
and facilitates the engineering of software. This synthesis technology is based upon Linear-Time-Logic (LTL) and is a declarative approach that permits the synthesis of a solution to the first mediation problem. In the second workshop it was verified that this is a complete solution.

"Abductive Synthesis of the Mediator Scenario with jABC and GEM" by Kubczak et al. is yet another use of the basic jABC technology but this time in combination with AI-style planning. The planning work was done at SAP research using a policy-based approach. The output of the planner was translated to the jABC platform, which performed all of the execution. The first mediation problem was completely solved.

"Data and Process Mediation Support for B2B Integration" by Zaremba et al. is a further extension of the WSMO technology for semantic web services. At this workshop, the team solved the very difficult payments mediation scenario using a fundamentally very flexible rule-based system. Inspection of the code revealed a few hard-coded elements that should be made more dynamic in future implementations, but the problem was verified to be solved.

The second Challenge workshop of 2008 was held at the International Semantic Web Conference in Karlsruhe, Germany (ISWC 2008). There were only two papers at this workshop since most of the time was spent on testing of the surprise problem, as documented on the Challenge wiki. We also completed the verification of jABC/LTL synthesis technology for the first mediation problem.

"Model-driven Service Integration using the COSMO Framework" by Quartel et al. described the solution to the first mediation problem, which was verified. The technology is very similar to that of the jABC technology of the Dortmund/Potsdam team. Future comparisons will be interesting.

"Advances in Solving the Mediator Scenario with jABC and jABC/GEM" by Lemcke et al. described the solution of the purchase order mediation and payment problems of the mediation scenario with jABC, confirming the adequacy of this graphical process-oriented model driven framework for business logic orchestration purposes, and its integration with the GEM planning approach, thus confirming the flexibility of this event calculus-based declarative approach, though it was clear from the code review that even more flexibility could have been introduced with more use of the fundamental AI-planning approach. In any case, the approaches were verified to solve all of the problems attempted and were two of several technologies that were successful in solving the surprise problem.

All of the papers in the two workshops, which are contained in this technical report, represent significant advances in the evaluation of claims of semantic web services technologies.

Acknowledgments: Omair Shafiq made a major contribution to the publishing of this technical report as well as workshops organization. The testbed and all verifications were performed by Srdjan Komazec. Federico Facca managed the surprise problem development and execution. Ulrich Küster contributed extensive time and though on the organization of the workshops and technical issues, especially in his role as Program Chair for the Karlsruhe workshop. The workshops received extensive crucial support from the Stanford Logic Group, SAP, STI Innsbruck and STI International.
SWSChallenge workshop at ESWC 2008
Debugging OWL Ontologies - A Reality Check

Heiner Stuckenschmidt

Universität Mannheim, Institut für Informatik,
A5, 6 68159 Mannheim, Germany
e-mail: heiner@informatik.uni-mannheim.de

Abstract. One of the arguments for choosing description logics as the basis for the Web Ontology Language is the ability to support the development of complex ontologies through logical reasoning. Recently, significant work has been done on developing advanced methods for debugging erroneous ontologies that go beyond the pure identification of inconsistencies. While the theory of these methods has been studied extensively little attention has been paid to the application of these Methods in practice. In this paper, we evaluate existing implementations of advanced methods for debugging description logic ontologies. We show that most existing systems suffer from serious problems with respect to scalability but surprisingly enough also with respect to the correctness of the results on certain test sets. We conclude that there is a need for further improvements of existing systems in particular with respect to bridging the existing gap between theory and practice of debugging ontologies.

1 Introduction

Description logics have been at the heart of semantic web technologies as a mechanism for formalizing ontologies. The ability to detect inconsistencies in ontology models is one of the most often quoted advantages of description logics and this argument has been extensively used to justify the use of description logics as a foundation for the web ontology language OWL. It has turned our very quickly, though, that being able to detect inconsistencies is only half the battle. Once an inconsistency has been found, providing an explanation of its origin and suggestions for resolving the inconsistency are equally important. This is the case because in complex or simply large ontologies it is often not feasible for the user to spot reason for an inconsistency. Providing support for this task is essential to the success of expressive ontology languages such as OWL because users will not be willing to use languages that are too complex for them to handle. A number of approaches have been proposed to deal with the problem of identifying the origins of problems and in description logic based ontologies and suggesting fixes for these problems. Early approaches contributing to the problem are methods for generating explanations of subsumption proofs in description logics that can be applied to the analysis of inconsistent concepts by explaining the subsumption relation of the respective concept and the empty concept. These approaches are normally based on the concepts of natural deduction, that is, they try to construct deductive consisting of a series of applications of deduction rules. These rules almost directly translate to an explanation in controlled language. More recently, a new line of research has evolved that applies principles of model-based diagnosis [9] to the problem. These approaches interpret inconsistencies as symptoms and use them to derive minimal sets of axioms...
that cause these symptoms thereby constituting a diagnosis for the problem. Existing approaches in this area can be distinguished into black-box approaches that use an external reasoner as a black box for testing whether certain combinations of axioms still cause an inconsistency and white-box approaches that modify the proof procedure of existing reasoners in such a way that debugging information is recorded during the reasoning process [7]. While the theoretical aspects of debugging ontologies have been investigated in details, there is relatively little experience regarding the performance of debugging methods in practice. Most of the methods have been applied to example ontologies individually to show their feasibility but so far no attempts towards a systematic evaluation of existing debugging tools have been reported in the literature. In this paper, we take a first step towards a systematic evaluation of debugging tools. In particular, we apply existing debugging tools to a benchmark dataset that has been created for this purpose and discuss the results of the different approaches. In particular, we look at the functionality of the tools, completeness and correctness and minimality of the debugging results as well as run-time needed. Based on the results of our experiments, we draw rather negative conclusion about the practical merits of currently existing debugging tools. In particular, we show that most systems have serious performance problems and often even produce incorrect results. Only one system seems to be ready for practical applications. The paper is structured as follows. We first recall some basic notions of description logics and the theory of debugging description logics.

2 Preliminaries

Before we take a closer look at existing tools for debugging ontologies, we first have to review some basic notions of description logics and the theory of debugging description logics.

2.1 Inconsistency, Unsatisfiability and Incoherence

Based on the model theoretic semantics of description logics (compare [1]), a number of formal properties have been defined that form the basis for any debugging effort. In particular, there are several properties of an ontology that indicate different kinds of conflicts in the formalization that have to be resolved by a debugging method.

The most basic property indicating a conflict in an ontology is inconsistency of the overall model. Inconsistency is an unwanted property, because inconsistent theories logically imply any fact thus making the definitions mostly useless in the context of logical reasoning.

Definition 1 (Inconsistency). A description logic terminology $T$ is said to be inconsistent if there is no interpretation $I$ that is a model for $T$.

While the notion of inconsistency exists for any kind of logic, there are weaker forms of conflicts that occur in the context of description logic. In particular, a terminology can only become inconsistent in the presence of an A-Box. In any case where
no A-Box exists, the interpretation mapping all concepts and relations to the empty-set is always a model of the terminology. A property that captures conflicts that solely exist in the terminology is the notion of concept unsatisfiability. A concept is said to be unsatisfiable if its definition does not allow a consistent instantiation.

**Definition 2 (Unsatisfiable Concept).** Let \( T \) be a terminology and \( C \) a concept in \( T \). \( C \) is said to be unsatisfiable if for all models \( I \) of \( T \) we have \( C^T = \emptyset \).

While unsatisfiable concepts to not make the ontology as such inconsistent still indicate potential problems, because adding instances to the ontology might result into an inconsistent model with all the unwanted consequences. As the role of ontologies in practical applications is mostly in to provide additional semantics for instance data, an ontology containing unsatisfiable concepts is often as bad as an inconsistent one. The existence of unsatisfiable concepts in an ontology is also called incoherence.

**Definition 3 (Incoherence).** Let \( T \) be a terminology. \( T \) is called incoherent if there is at least one concept \( C \) in \( T \) and \( C \) is unsatisfiable.

### 2.2 Debugging Description Logics

In [10] Schlobach and others provide some basic definitions that are commonly seen as the basis for ontology debugging. These definitions are partly taken from earlier work on general diagnosis and have been adopted to the need of description logic terminologies as a target for the diagnosis. The corresponding theory takes unsatisfiable concepts as a starting point and provides a number of definitions that correspond to subtasks in the debugging process. The first important concept is that of a minimal unsatisfiability preserving Subterminology (MUPS). Informally a MUPS is a minimal set of concept definitions from a terminology that together make a concept unsatisfiable.

**Definition 4 (MUPS).** Let \( T \) be a terminology. A Terminology \( T' \subseteq T \) is a MUPS for a concept \( C \in T \) if \( C \) is unsatisfiable wrt. \( T' \) and \( C \) is satisfiable wrt. all terminologies \( T'' \subseteq T' \).

MUPS roughly correspond to the notion of conflict set in classical diagnosis and computing MUPS for a given unsatisfiable concept is one of the most basis tasks for any debugging system as the MUPS can be assumed to contain at least one wrong definition that needs to be corrected or removed to fix the conflict in the ontology.

MUPS provide a basis for computing minimal incoherence preserving subterminologies (MIPS). MIPS are minimal sets of concept definitions that make any of the concepts unsatisfiable. MIPS are interesting, because they correspond to minimal conflict sets with respect to the problem of incoherence.

**Definition 5 (MIPS).** Let \( T \) be a terminology. A Terminology \( T' \subseteq T \) is a MIPS for \( T \) if \( T' \) is incoherent and all \( T'' \subseteq T' \) are coherent.

MUPS and MIPS provide a basis for computing minimal diagnosis for the problems (unsatisfiability or incoherence respectively). In particular, the hitting set algorithm originally proposed by Reiter [9] can be used to compute minimal diagnosis from both MIPS and MUPS. A minimal diagnosis, in this case is defined as a minimal set of concept definitions that if removed from the ontology solves the conflict at hand.
Definition 6 (Diagnosis). Let $T$ be an incoherent terminology. A diagnosis for the incoherence problem of $T$ is a minimal set of axioms $T'$ such that $T - T'$ is coherent. Similarly, a diagnosis for unsatisfiability of a single concept $C$ in $T$ is a minimal set of axioms $T'$, such that $C$ is satisfiable with respect to $T - T'$.

As computing diagnosis is very expensive computationally, Schlobach and others propose to use approximate notions of diagnoses called pinpoints. We will not go into details here but just think in terms of possibly approximate diagnoses.

3 Debugging Systems

In our evaluation we used ontology debugging tools that are freely available on the web. We can distinguish between pure debugging systems that read an ontology in a standardized format and computes certain features such as MUPS, MIPS and approximate diagnoses and ontology editors with an integrated debugging facility. In the following, we briefly describe the systems included in our experiments.

**MUPSter** The MUPSter System is an experimental implementation of the white-box debugging method described in [10], in particular for computing MUPS, MIPS and approximate diagnoses. It was rather meant as a proof of concept than a debugging tool for real use. The System has a number of limitations connected with the expressiveness of the ontologies it is able to handle. In particular, the MUPSter System is only guaranteed to work on unfoldable Terminologies in the logic $\mathcal{ALC}$. Further, the system does not directly work on OWL. It requires input ontologies to be encoded using the KRSS or the DIG format for description logics. We nevertheless included the system in our evaluation in order to test what the impact of these limitations are in practice. The version of the MUPSter System used is available online\(^1\).

**DION** The DION System is an implementation of black-box debugging that has been developed in the context of the SEKT project [\]. The system is completely implemented in SWI-PROLOG and uses the DIG interface for communicating with an external reasoner. In our experiments, we used version 1.7.14 of the RACER system as an external reasoner for all black-box systems to guarantee fairness. Similar to the MUPSter System, DION works on ontologies in the DIG format and as MUPSter computes MUPS, MIPS and approximate diagnoses. The tool comes with a translation function for converting OWL ontologies to the DIG format that we used for generating input data for MUPSter and DION. The version of DION used in the evaluation is available online\(^2\).

**RADON** The RADON system that implements a debugging method that is inspired by belief revision. The system uses a weakening based revision function that basically removes certain axioms from the model such that the conflict is resolved and the amount of information removed is minimal with respect to some measure of minimality [8]. The axioms to be deleted correspond to MUPS or MIPS respectively. This general approach allows the system to debug incoherent as well as inconsistent ontologies. The system uses the KAON 2 Reasoner for checking consistency and coherence of ontologies and

---

\(^1\) http://www.few.vu.nl/schlobac/software.html

\(^2\) http://wasp.cs.vu.nl/sekt/dion/
can therefore be seen as a white-box approach. A special feature of the system, that is not used in this evaluation, however, is the ability to add knowledge to an existing model revise the existing model based on the new information. The additional information that the new information is always correct helps to restrict the search space and can therefore be assumed to improve the debugging result. The version of RADON used in this evaluation is available online3.

Protege  Being the most often used ontology editor around, some effort has been spent on extending Protege with a number of mechanisms for supporting the user in building correct models. One of these mechanisms is a debugging facility that has been developed in the context of the CO-ODE project [12]. In contrast to the systems mentioned so far, the method does not rely on a sound formal basis in terms of diagnostic reasoning or belief revision. Instead Protege uses a set of heuristic rules that are applied to the description of an unsatisfiable class and derive possible reasons for the unsatisfiability based on certain modeling patterns. The generated explanation does not always correspond to a MUPS but it can also contain natural language explanations such as 'The superclass is unsatisfiable'. As a results, the method is not complete. The authors argue, however, that this is not a drawback in practical situations as most common modeling mistakes are covered by the rules. The debugging system we used in the evaluation is part of Protege 4.0 which is available online4.

SWOOP  The SWOOP editor developed at the University of Maryland also has an integrated debugging mechanism. It benefits from the tight integration with the Pellet reasoner that has been extended with white-box methods for debugging unsatisfiable concepts [5]. The mechanism used in the evaluation is the general explanation function that can be used to generate explanations for any derived subsumption statement including subsumption by the empty concept. The explanation is generated on the basis of a trace of the tableaux proof and consists of a number of axioms that correspond to a MUPS for the respective concept. The version of the debugging method used in this evaluation is contained in SWOOP version 2.3 and can be obtained online5.

4 Experiments

The goal of our experiments was to find out how well the different systems cope with realistic debugging tasks at different levels of difficulty. For this purpose, we created a benchmark dataset consisting of ontologies about the same domain that have been designed by different people. We introduced inconsistencies in these ontologies using automatic matching systems for conducting a pairwise integration of these ontologies. The resulting merged ontologies contain unsatisfiable concepts which are a result of known errors in the matching process that have been documented before. This has the advantage that despite not dealing with toy examples we have a pretty good idea of which axioms are the source of the problem and should be detected by the debugging systems. In the following we describe the data used as well as the experimental setting and the results in more details.

3 http://radon.ontoware.org/  
4 http://protege.stanford.edu/  
5 http://code.google.com/p/swoop/
4.1 Datasets

We evaluated the debugging systems using the OntoFarm Dataset. It consists of a set of ontologies in the domain of conference organization that have been collected by researchers of the Knowledge Engineering Group at the University of Economics Prague [11]. The ontologies have been built manually by different persons some based on experiences with organizing conferences. Some are based on existing conference management tools. In general they cover the structure of a conference, involved actors, as well as issues connected with the submission and review process.

We used the automatic matching systems falcon [4] and CTXmatch [2] to compute hypotheses for equivalence mappings between pairs of these ontologies. We translated the mapping hypotheses into equivalence statements and created new ontologies consisting of the union of the mapped ontologies and the equivalence statements. Due to mistakes in the matching process, these union ontologies often contain unsatisfiable concepts. Based on the evaluation of the mappings carried out in the context of the Ontology Alignment Evaluation Initiative [3] we know exactly which axioms are erroneous and should ideally be detected by the debugging systems.

4.2 Concrete Test Cases

From the data described above, we chose a number of test cases. These test cases correspond to those combinations of ontologies using mappings of the two systems mentioned above that contain unsatisfiable concepts. It turned out that these contain test cases of different levels of complexity ranging from very simple tests that almost all systems could manage to very complex ones only some systems solved correctly. As indicators for the complexity of a debugging problem we used the following criteria

1. The size of the ontology in terms of number of classes
2. the number of unsatisfiable concepts in the ontology
3. the average number of MUPS per unsatisfiable concept
4. the kinds of axioms that occur in a MUPS

All of these criteria have an impact on the difficulty of the problem as they contribute to the size of the combinatorial problems that has to be solved at the different levels of the debugging process. According to these criteria we distinguish between simple, medium and hard problems. In total, we used the following test cases three of which we classified as easy, three as medium and two as hard tasks.

4.3 Experimental Setting

We first used each of the debugging tools in the evaluation to check the test data sets for unsatisfiable concepts. This may seem to be a trivial, but it turned out that some of the tools already had problems finding all unsatisfiable concepts due to limitations on subset of OWL constructs the tool handles correctly. We then used each tool to compute MUPS for all the unsatisfiable concepts found and compared the results to each other. In contrast to the first step, we were not able to compare the results with a provably correct result. We rather compared the results of the different tools and did a manual

---

inspection of the results proposed. In some cases, we tested correctness and minimality of results by systematically removing axioms from the ontology an rerunning the test. In this way we could show that some of the results are not minimal (they still lead to unsatisfiability after axioms had been removed from the MUPS) and some are not even correct (after fixing the known problem, the tool still computed a MUPS for a conflict that did not exist any more). We also explored more advanced features of the tools in a more ad hoc fashion as a systematic evaluation of MIPS and diagnosis was impossible due to the extreme run-times that often forced us to interrupt the computation after a couple of hours.

5 Results

The results of our experiments were not very encouraging and revealed a number of problems some of the tools face when being applied to unforeseen tasks.

5.1 Performance

Our experiments confirmed the observation that black-box approaches for debugging suffer from serious performance problems due to the combinatorial explosion of reasoner calls needed for computing the diagnosis. The Black-Box Systems in the test, namely RADON and DION suffered from very long run times. In the case of DION we faced serious performance problems with the result that only the smallest test case could be solved by the system. In the second case the computer ran out of memory. All other test where interrupted after 12 hours(!). This negative result can probably be explained by the use of PROLOG as an implementation language that is known to be rather resource consuming and by the inherent complexity of the black-box approach the system implements. Acceptable performance was observed by Protege and SWOOP which is not surprising as these Systems aim at supporting user interaction which requires real-time behavior.

5.2 Detecting Unsatisfiable Concepts

The most surprising insight for us was that some of the tools already have problems with the very basic task of finding unsatisfiable concepts which we assumed to be a solved problem. As table 1 shows the only tool that correctly identified all unsatisfiable concepts was SWOOP while Protege and MUPStor missed some of the problems. In the case of MUPStor this was expected as the tool only covers a sublanguage of OWL. Looking at the impact of this restriction it turns out that the system still works for most simple cases and still finds most of the unsatisfiable concepts in the complex ones. Protege seems to lose information in the communication between the editor and the external reasoner. So far we did not find out what exactly is the problem. Possible problems are with the use of datatype properties in definitions and nested restrictions that sometimes seem to get lost. Even more surprising is the fact that the RADON system overshoots and marks concepts as unsatisfiable that are not. A closer look revealed that the System does not correctly interpret datatype properties. As a consequence, all concepts whose definitions involved datatype properties were named unsatisfiable which lead to the surprising result.
### 5.3 Coverage of the Debugging Task

The functionality of the debugging tools used differs, in order to be able to compare them, we use the formal definitions of MUPS, MIPS and diagnosis as a basis of our comparison. It turns out that all systems compute some kind of MUPS - in the case of Protege the result of the debugging process can be seen as an approximation of a single MUPS. All other systems compute all MUPS. MIPS are only computed by the pure debugging systems in the evaluation and (approximate) diagnoses are only provided by MUPSter and DION. We therefore use the MUPS as the main basis for our comparison.

### 5.4 Computing MUPS

The actual debugging functionality of the tools was hard to compare as we do not have provably correct results for the example ontologies. A manual inspection of the debugging output, however, revealed some fallacies of the tools. While the restriction of the MUPSter tool to a subset of OWL only led to a small degradation of the results with respect to finding unsatisfiable concepts, the tool only produced trivial MUPS. In particular, the MUPS only contained the bottom concept. This means that it is not applicable to more expressive ontologies. The DION tool computed a correct MUPS and even MIPS and an approximate diagnosis for Test 1 but as mentioned above failed on the other tests due to memory and runtime problems. The PROTON Tool only worked correctly on some of the tests, on others, in particular Test 1 the MUPS computed by PROTON were neither minimal nor unsatisfiability preserving. For simple problems that only had a single MUPS the tool came up with more than ten different MUPS most of which contained almost the entire ontology. This can probably be explained by the problems with determining unsatisfiability which for the case of a black-box approach completely messes up the revision. With respect to the heuristic debugging approach implemented in Protege we detected two major problems. The first is the creation of cyclic explanations for equivalent concepts. Quite often, the tool explains the unsatisfiability of a concept with the unsatisfiability of another equivalent concept and vice versa without pointing to the actual problem which resides in the definitions of these concepts. Further, we found errors in the definition of the heuristic rules. In particular, there were cases where domain and range definitions were mixed up leading to wrong explanations. We did not check all the debugging rules for correctness but there could be more problems like this. The only tool that seems to correctly compute MUPS for all of the test cases was the SWOOP system, although, as mentioned above we did not formally check the correctness and minimality of all the results.
6 Conclusions

The conclusion we have to draw from our experiments is that most existing approaches for debugging OWL ontologies are mostly of theoretical interest and that more work is needed to make these methods applicable in practice. The lessons learned from the evaluation concern the following three points:

Well-foundedness The results of the heuristic debugging approach implemented in Protege show the need for the use of theoretically well founded methods as a basis for ontology debugging. While the authors of the method argue that it covers most of the relevant cases, we found out that if applied outside there cases, which in the case of Protege are modeling errors, the system’s performance is rather poor. In our test cases the errors are rather matching than modeling errors. As a consequence, unsatisfiable concepts almost always had an equivalent concept, a situation Protege cannot deal with correctly. Our experiments also clearly shows the problem of heuristic approaches in terms of correctness. The error we spotted only turned up when the tool was used in an unforeseen situation. If heuristic methods are used, more systematic testing is required to exclude these kinds of problems.

Robustness To our surprise also well-founded methods produced wrong results. In our cases this was a result of the inability of the underlying reasoning services to correctly interpret the complete OWL standard. What we missed in this context was a graceful degradation of the system’s performance. In most cases, the inability to interpret a certain definition led to a completely wrong or trivial result. An interesting topic for future research is in methods with better robustness properties, i.e. methods that do not completely fail on input they cannot handle completely but still compute diagnoses that are correct and minimal with respect to the subset understood by the system. On a more technical level, existing systems should pay more attention to non-logical features of OWL, in particular to the use of datatype properties. These seem to be the main source of practical problems encountered in our experiments.

Performance Run-time performance is a major issue in the context of a practical application of debugging methods, especially because debugging will normally be an interactive process. Our experiments confirmed the common knowledge that black-box approaches suffer from their computational complexity as they normally require an exponential number of calls to an external description logic reasoner. None of the black-box systems in the test showed a performance that would allow for an interactive debugging strategy. Another interesting observation is that many black-box methods discussed in the literature are not available as tool, probably for this reason. The experiments also showed that besides the theoretical complexity, the implementation strategy is also an issue. The poor performance of the DION System clearly shows that the PROLOG-based solution is only useful as a proof of concept, but not ready for real applications. Not surprisingly, the two debugging tools integrated in ontology editors did not use black-box approaches.

In summary, the way to move forward is to further develop white-box approaches for ontology debugging focusing on the robustness of the methods and the coverage of the OWL Standard. Further, there is a case for specialized debugging tools such
as the one reported in [6] that addresses the specific problem of debugging mappings between ontologies and therefore shows a much better performance on the test datasets, mainly because it restricts the search space to axioms representing mappings while the ontologies themselves are assumed to be correct. Such a mechanism allowing the user to focus the debugging process to a subpart of the ontology would clearly enhance existing approaches.

References

Appendix A: Test Cases

Test 1 (easy)
Ontologies  CRS and PCS, matched with CTXmatch
Size  Small (39 Concepts)
Problems  A few (2 Unsatisfiable Concepts)
Complexity  low (one MUPS per Concept, only subsumption and negation involved)

Test 2 (easy)
Ontologies  CRS and SIGKDD, matched with falcon
Size  Small (48 Concepts)
Problems  A few (2 Unsatisfiable Concepts)
Complexity  low (one MUPS per Concept, only subsumption and negation involved)

Test 3 (easy)
Ontologies  CRS and CONFTOOL, matched with falcon
Size  Medium (64 Concepts)
Problems  A few (2 Unsatisfiable Concepts)
Complexity  low (one MUPS per Concept, only subsumption and negation involved)

Test 4 (medium)
Ontologies  CRS and CMT, matched with CTXmatch
Size  Small (44 Concepts)
Problems  One (1 Unsatisfiable Concepts)
Complexity  medium (one MUPS per Concept, subsumption, equivalence, union and negation involved)

Test 5 (medium)
Ontologies  CONFTOOL and EKAW, matched by falcon
Size  Larger (112 Concepts)
Problems  Some (7 Unsatisfiable Concepts)
Complexity  Medium (1-5 MUPS per concept, only subsumption and negation involved)

Test 6 (medium)
Ontologies  SIGKDD and EKAW, matched by falcon
Size  Larger (123 Concepts)
Problems  Some (4 Unsatisfiable Concepts)
Complexity  Medium (2 MUPS per Concept, subsumption, negation and domain restrictions involved)

Test 7 (hard)
Ontologies  CMT and CONFTOOL, matched by CTXmatch
Size  Medium (68 Concepts)
Problems  Many (18 Unsatisfiable Concepts)
Complexity  High (2 to 5 MUPS per Concept, subsumption, negation, domain restrictions, inverse and number restrictions involved)
Test 8 (hard)

**Ontologies**  CONFTOOL and EKAW, matched by CTXmatch

**Size**  larger (112 Concepts)

**Problems**  a lot (39 Unsatisfiable Concepts)

**Complexity**  high (2 to 6 MUPS per Concept, subsumption, negation, number restrictions, domain restrictions, inverse and existential quantifiers involved)
Semantic System Integration – Incorporating Rule-based Semantic Bridges into BPEL Processes

Nils Barnickel, Ralf Weinand, Matthias Fluegge
Fraunhofer Institute for Open Communication Systems (FOKUS),
Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany

Abstract. This paper describes how semantic bridges realized in terms of rule-based ontology mappings can be incorporated into BPEL processes. The approach is explained by applying it to a semantic system integration scenario in the eBusiness domain defined as the “purchase order mediation” scenario in the context of the Semantic Web Service Challenge. The presented approach relies strongly on the existing Web standards and is based on widely adopted open source software components.

Keywords: semantic interoperability, semantic mediation, semantic bridge, ontology mapping, SWRL, BPEL, SWS-Challenge

1 Introduction

The advent of the service-oriented architecture (SOA) model and its implementation in the form of Web services has contributed significantly to facilitate the technical integration of information systems. However, semantic interoperability, i.e. the semantically sound exchange of data between heterogeneous systems, still represents a challenging task. The main reason is that domain- and application-specific requirements have produced and will always produce different information models for one and the same problem.

The alignment of heterogeneous information models is impeded by the fact that they are often represented in a semantically poor manner, focusing only on structural specifications for data exchange. This makes system integration mainly a manual effort and it requires considerable technical skills to define appropriate syntactical transformations. The application of Semantic Web technologies to system integration problems promises to mitigate this problem since the formal definition of semantics paves the way for (semi-)automated data and process mediation. Moreover, shifting information integration from the structural to the conceptual level represents a further step towards the ultimate goal of SOA, namely to align business and IT.

1.1 Challenges and basic approach

A light-weight and effective semantic system integration approach is the use of semantic bridges as described in [1]. The basic idea is to wrap existing information resources with semantically described Web services and to leverage rule-based ontology mappings in order to achieve interoperability and to reduce manual efforts in the composition and execution of heterogeneous Web services. The paper at hand will briefly present how this approach can be applied to implement the “purchase order mediation” scenario as defined in the context of the Semantic Web Service Challenge (SWSC).

The interacting systems of the SWSC mediation scenario mainly differ with regard to the following aspects

1. data formats (i.e. granularity and denotation of data elements) and
2. interaction patterns (order and granularity of operations)

The approach presented in this paper will address the first issue by applying semantic bridges to mediate between different information models and representations. The second issue will be addressed by using the Business Process Execution Language (BPEL) and an appropriate BPEL-compliant execution engine to orchestrate the services provided by both parties. This paper does not cover goal-oriented plan creation (compared to other approaches such as WSMO [2] or SWSF [3]) and leaves the planning task (i.e. which services to include at which part into the composition) to the business domain process expert (cf. section 1.2 Related Work). Thus, this paper presents a lightweight approach to reduce manual semantic mediation efforts by integrating semantic bridges into BPEL.

Semantic bridges describe the relations between entities in business information models that are defined in different ontologies but have a similar meaning. At the same time semantic bridges define appropriate mappings in order to translate instances of such entities or so called concepts. Ideally such a mapping can be included directly and transparently in the reasoning processes, which allows for drawing conclusions and thus provides the foundation for tool-supported semi-automatic semantic mediation.

The core concept of semantic bridges lies in the shift of semantic mediation from the structural to the conceptual abstraction level in order to reduce efforts for achieving semantic interoperability. Moreover, semantic bridges cannot just be applied in the execution phase but also in the design phase of a business process. A matching engine transparently applies semantic bridges and performs the reasoning over semantically described relationships (such as inheritance or equality between concepts), thus enabling a composition tool to semi-automatically support the design of interaction patterns by issuing recommendations for suitable assignments between output and input parameters of different Web services. Consequently, achieving semantic interoperability requires less manual integration efforts.

A promising approach to meet the described requirements is the use of expressive rule languages to capture mappings and to enable the direct application of the specified mappings for corresponding instance transformations. Logic-based rules are computationally complete. Hence, by defining semantic bridges in terms of logic-

---

based rules, any kind of mapping relation (one-to-one, one-to-many, many-to-many) can be described. The absence of technical transformation code increases ease and maintainability of the semantic bridges. Furthermore and most importantly, an inference service can directly apply the rules as part of its reasoning process, i.e. the transformation of concept instances and their correct classification as well as potential further conclusions are handled in a well-integrated manner. When applying the approach in combination with existing ontology mapping tools [4,5] which allow to semi-automatically define the mappings as semantic bridge rules, manual integration efforts can be reduced substantially.

It has been recognized that the success of Semantic Web technologies relies on the reuse and integration of existing Web standards. The most widely-used standard for the composition of Web services is BPEL. A considerable number of mature BPEL-compliant process execution engines testify the broad industrial support for this standard which provides a rich set of control and data flow constructs for defining and aligning the interactions between the participating actors in a business process. The solution outlined in this paper raises the claim of being not only of theoretical but also of practical relevance. Consequently, the approach described in [1] was extended towards semi-automated data mediation in Web service processes that are formalized in terms of the BPEL standard.

The main challenge on this regard is to find a suitable mapping between different abstraction levels: While at design time ontologies and rules are used for data representation, data flow and mediation, BPEL execution engines make use of hierarchically structured XML Schema types, XPath and XSLT transformations. In order to face this challenge, the starting point is to exploit the RDF/XML serialization of ontologies for data representation on the BPEL level. Furthermore, BPEL enhancements have to be developed to integrate semantic bridges and to support data flow specifications in terms of rules. These enhancements are implemented as external functions that can be plugged into BPEL engines using a standardized extension mechanism as described in more detail in section 2. Also the application of rule-based semantic bridges and their integration into the data flow will be illustrated in section 2 for the “purchase order mediation” scenario.

1.2 Related Work

The long term vision behind Semantic Web services is to enable dynamic goal-oriented service composition and to use powerful inference engines and matchmaking mechanisms in order to automate the whole composition process including discovery, composition, execution and interoperation of Web services. Research background comes from the Semantic Web community on the one hand and from the field of dynamic planning in artificial intelligence research on the other hand. Significant work has been done in the context of WSMO [2]. WSMO includes a mediator concept to deal with the interoperation problems between Semantic Web services. The approach considers specific mediator services which perform translations between ontologies. These mediators attempt to reconcile the differences between goals of Web services. In the context of the SWS-Challenge this approach has been applied in [6]. The approach presented in this paper does not cover goal-oriented plan creation
as it intentionally leaves the planning task (i.e. which services to include at which part into the composition) to the business domain process expert. Consequently, as presented in section 1.1, a more lightweight approach to semantic mediation has been developed.

The core concept of this approach is the integration of ontology mappings into BPEL processes. The ontology mappings are described through a set of description logic-based bridging axioms which refer to concepts or properties of a source ontology and specify how to express them in a target ontology. Thus, bridging axioms can be realized as rules which describe the transformation. The advantage of this rule-based approach is that reasoning over the described mappings can be applied straightforward as the transformation rules can be integrated into the regular ontology inference process, e.g. classifying etc. In particular, the presented approach in this paper applies SWRL rules, which are broadly supported by available Semantic Web frameworks and it makes use of the facet analysis classification mechanism (cf. section 2 Scenario Realization) supported by standard OWL semantics.

2 Scenario Realization

In order to be able to apply our mediation concept to the “purchase order mediation” scenario provided by the SWSC workshop we have to introduce several conceptual components which are numbered from one to five as illustrated in Fig. 1.

In the following section the conceptual components for semantic mediation and their realization based on widely adopted open source products is described in detail.
2.1 Heterogeneous Domain Ontologies

For our scenario solution we assume that two different ontologies have been developed by RosettaNet domain experts and independently by domain experts of the Moon legacy system. The approach of multiple independent ontologies on the domain level takes into account the realistic point of view that existence of a single global ontology can not be assumed to cover all autonomous actors and organizations across various domains in a system integration scenario. Fig. 2 shows an outline of these heterogeneous information models which are formalized in the OWL [7] ontology language using defined classes. The information models differ in their semantic subgraph. As the concept Partner in the RosettaNet ontology is defined in terms of three object properties a semantically corresponding concept Customer in the Moon ontology just features two object properties containing the same information, however defined at a lower level at granularity. By modeling these concepts as defined classes, corresponding OWL individuals can be easily classified by a reasoner supporting OWL facet analysis.

![Fig. 2. Heterogeneous ontologies representing the different information models of the Blue system (RosettaNet) and the Moon system (legacy).]

The existing XSD-based messages such as the PIP3A4 PO request message that is sent to the mediator by the Blue system are lifted to the concepts described in these heterogeneous domain ontologies using Semantic Web services, which are described in the following.

2.2 Semantic Web Services

The Web services provided in the scenario are annotated with concepts from the domain ontologies described above by applying the OWL-S ontology [8] for realizing Semantic Web services. Input and output parameters of Web services are linked to ontology concepts. For instance, in the scenario the Semantic Web service for the Moon CRM is expecting the defined class Customer which – among others – defines the property hasBusinessName. This property is used as the search criteria for the customer lookup. If a customer with the given name is found, an IdentifiedCustomer OWL individual is returned containing all customer attributes supplied by the CRM system. Furthermore, lifting and lowering definitions for converting the incoming and outgoing XSD instances to OWL instances are defined. This so called grounding
mechanism in OWL-S is based on XSL transformations which are supposed to be developed by domain experts who semantically enrich their existing portfolio of Web services. OWL-S provides the main advantage in terms of tool support compared to other light weight approaches focusing mainly on service input and output annotations. In this regard the otherwise well fitting candidate SAWSDL [9] has not been chosen as a Java-API that fully executes Semantic Web services is not yet available. Therefore, the Mindswap OWL-S API has been applied to get programmatic access to read, execute and write OWL-S service descriptions. However, XSLT does not work on the same data model as OWL and thus can only operate on the OWL serialization. In our context the RDF/XML-ABBREV serialization applied in the OWL-S API implementation does not allow to exploit the full potential of polymorphism. When a polymorph individual is serialized using the RDF/XML-ABBREV format one of the types it holds is non-deterministically selected and the last fragment of the types URI is taken for the opening tag for this individual’s XML serialization. The other types are expressed by separate <rdf:type.../> sub elements. This varying structure complicates the development of XSLT code dramatically. To overcome this weakness, the OWL-S API, has been adjusted accordingly. Now internally the basic RDF/XML serialization is applied. This means that all types are represented equally as sub elements, which allows to define straighter XSL transformations. Hence, the mapping from polymorph OWL serializations to single typed XML Schema instances can be achieved in terms of XSLT rules that match exactly the type which has been defined in the OWL-S input description.

The processing of the Web service results is less complicated as the received message parts correspond to the data model XSLT was designed for. XSLT rules can easily match the XML Schema instances and fill predefined skeletons of serializations of OWL individuals. It has to be mentioned that further modifications of the OWL-S API were necessary in order to provide support for WSDL messages typed via XML-elements and for operation calls based on Document/Literal communication as used in the provided Web services of the scenario.

### 2.3 Semantic Bridges

Obviously the Partner and Customer concepts presented above cannot be exchanged between communicating partners by default, although they represent the same conceptual idea. In order to mediate between these concepts the domain experts of both domains have to define mappings – so called semantic bridges – between the two heterogeneous ontologies. These semantic bridges are formalized as rules based on the Semantic Web Rule Language (SWRL) [10] and are made publicly available.

The application of rule-based semantic bridges for semantic mediation can be illustrated based on the simple example of two ontology concepts (Partner and Customer) which, although representing intuitively the same concept, have been defined independently in separate ontologies (cp. Fig. 2) and hence differ in their semantic sub-graph.
By applying the semantic bridge rules, an instance of type Partner is furnished with additional properties e.g. with hasCustomerInfo combining the values of the BusinessDescription and the ContactInformation properties hasBusinessName and hasEmailAddress as illustrated in Fig. 3.

Having the class definitions on hand, a reasoner is now able to classify the instance as a member of the defined class Customer, since all required properties (including hasCustomerInfo) are present. Thus, within the scope of the mediation process any service, independently to which domain it belongs, can now make use of this instance as it is polymorph of type Partner and Customer, i.e. semantic interoperability has been established.

Consequently, the transformed and reclassified individual can now be used as input for querying the Moon CRM. Thus, by applying the semantic bridge, the heterogeneous data formats of the Blue system and the Moon system have been successfully mediated. Using traditional XML-based Web services without semantic annotations, such an approach would not be feasible. As it has been argued in [11], the static type bindings do not allow for effective polymorphism; in particular XML Schema lacks inheritance relations to be exploitable for matching.

Technically, a semantic bridge is integrated into the BPEL process by using a standardized extension mechanism in terms of a custom XPath function call within an
Assign-Copy activity. The custom XPath function is implemented as a Java component based on the Jaxen framework and thus can be integrated in any BPEL execution engine that supports the configuration of custom XPath functions. Internally, the Java component that executes the semantic bridge relies on the Protégé-OWL API, which depends on the OWLAPI, Jena and Pellet for handling ontologies and performing DL-reasoning and on the Jess-rules engine which executes the SWRL rules [12].

2.4 Rule-based Data Flow

The usage of SWRL rules to mediate between heterogeneous ontology concepts is similarly applied in order to express the dataflow within the BPEL process. We distinguish between rules defining a semantic bridge and rules defining data flow within the process. Semantic bridges are developed on an ontology level independently of actual application scenarios. Hence, they can be reused for various integration scenarios between the involved parties. Rules defining the data flow are included into the BPEL process at design time of the integration scenario. In our scenario realization the data flow rules have been defined manually. However, as described in section 3 and in [1] the approach of using description logic-based Semantic Web services provides the foundation to generate the data flow in a semi-automatic manner. The concept of rule-based data flow is illustrated in Fig. 4.

The illustrated example shows the rule-based data flow integrated into the BPEL process in order to realize the hidden assumption of the purchase order mediation scenario: Although the search for a customer in the CRM system provides the mediator already with information such as AddressInfo or ContactInfo, still the information that is provided in RosettaNet messages should be used instead.

The first rule illustrated in Fig 4. binds the polymorph Partner/Customer individual holding the RosettaNet-based PhysicalAddress and ContactInformation, etc. Subsequently, the rule attaches the individual to a newly generated NewOrderRequest individual that is created in terms of SWRL built-Ins generated NewOrderRequest individual. The NewOrderRequest individual acts as the umbrella container for the input parameters of the Moon Create/NewOrder Semantic Web service.

The second rule attaches the hasCustomerID property received from the Moon CustomerLookupResponse Semantic Web service to the afore assigned Customer individual. Taking into account the corresponding OWL class definition, the individual is then classified as an IdentifiedCustomer. Thus, the above described assumption is anticipated and the IdentifiedCustomer individual can be used properly in the further process flow.
As already outlined above, the harmonization of the interaction patterns of both systems is achieved by means of a BPEL process definition and a corresponding BPEL engine [13].

The following example demonstrates how BPEL is combined with the rule-based data flow in order to mediate between different granularity levels in service calls. In the given scenario the `OrderItems` provided by the Blue system are aggregated in a single incoming call. However, the Moon system requires fine granular calls of the `AddLineItem` Semantic Web service, i.e. one service call for each single `LineItem`. Fig. 5 illustrates the corresponding BPEL-process part.
Fig. 5. BPEL-process part implementing the split of aggregated *OrderItems* and the invocation of the *AddLineItem* Semantic Web service provided by the Moon Order Management System.

The *ForEachItem*-loop matches each single *OrderItem* and extracts its URI from the XML serialization defined according to the RosettaNet ontology. The following XPath expression is used for this purpose.

\[
\text{($processInputLifted//rdf:Description/rdf:type[@rdf:resource="MoonOntology.owl#OrderItem"])[Scounter]/../@rdf:about}
\]

Subsequently, the URI is passed to the description logic based data flow component, which performs the actual data flow to the input variable of the Moon order management system, i.e. to the *AddLineItem* Semantic Web service. The URI of the *OrderItem* was chosen as the connecting conceptual element between the XSD-based data model of the BPEL process and the description logic based data model of OWL individuals and SWRL rules. Fig 6. illustrates the above described data flow rule.

Fig. 6. Rule-based data flow definition preparing the input of the Moon order management system, i.e. of the *AddLineItem* Semantic Web service. The variable *?itemURI* is replaced by the actual URI of the *OrderItem* extracted by the XPath expression at runtime.

The enhancements for applying the rule-based data flow and the Semantic Web service call have been implemented using the same standard BPEL extension mechanism as explained above.
3 Conclusion, Potential and Limitations

The approach presented in this paper aims at reducing the complexity of semantic mediation in order to facilitate system integration in a service-oriented architecture landscape. Semantic Web technologies have been used as an additional layer on top of existing WSDL-based Web services and XML Schema based message formats. The heterogeneous information models of the Blue system (RosettaNet) and the Moon system have been additionally expressed in terms of two autonomously developed domain ontologies. These domain ontologies have been combined with specific upper ontologies for Web services (OWL-S) in order to realize the Semantic Web services. Future versions of the presented implementation will be based on SAWSDL service descriptions. A corresponding Java API for the execution of SAWSDL Semantic Web services is currently under development by the authors of this paper.

The approach of multiple independent ontologies on the domain level takes into account the realistic point of view that in a system integration scenario the availability of a single global ontology covering the needs of all autonomous actors and organizations across various domains cannot be assumed. The well established paradigm of loose coupling is reflected on the semantic level by using independently evolving information models (ontologies) that are loosely coupled through semantic bridges.

While on the one hand semantic bridges target the semantic mediation challenge of heterogeneous data formats, the semantic mediation challenge of heterogeneous interaction patterns has been addressed by using a BPEL engine as the coordinating entity. The integration of Semantic Web services, semantic bridges and rule-based dataflow into BPEL processes thus provides the foundation for powerful tool support in semantic system integration scenarios. The description logic-based data model provided by ontologies in conjunction with semantic bridges allows for applying automatic semantic matching mechanisms based on polymorph representations of service parameters. In [1] this approach has been demonstrated in order to provide semantics-based tool support for Web service orchestration: Semantically matching service parameter parts are presented as assignment recommendations to the process expert, thus facilitating the data flow design. Based on the selection of the involved Web services and of the assignment recommendations an execution plan for the Semantic Web service composition can be constructed in a semi-automated manner. By applying the mechanisms described in this paper the generated process can be expressed in BPEL and executed by appropriate industrial mature workflow engines.

Thus, the presented approach relies strongly on the existing Web standards BPEL and OWL and thus raises the claim of being not only of theoretical but also of practical relevance. The main conceptual advantage of the presented approach is that semantic interoperability is addressed on the level of domain standards assumed as given in terms of ontologies. Thus, mediation between different representation formats in overlapping conceptualizations is only done once instead of performing it repeatedly on the application level during Web service composition. Consequently, the process expert can focus on process specific concerns and can leave the task of semantic mediation between heterogeneous information models to the domain experts. The task of semantic mediation between different interaction patterns is
supported by providing semi-automatic tool support for data flow design. Thus, the complexity in semantic system integration can be reduced substantially.

4 Future Work

Future work will focus on exploiting the introduced semantic layer (Semantic Web services, semantic bridges, rule-based data flow) for further extension of tool support in semantic system integration scenarios in the context of service oriented architecture landscapes. The long-term vision on this regard is to reduce technical complexity for process experts in system integration and thus fulfill the conceptual promise of service oriented architectures namely to enable sound alignment of business with information technology.

Acknowledgments. This work is partially supported by the European Commission through the research project QualiPSo (Quality Platform for Open Source Software, contract IST-FP6-IP-034763). The paper was developed in the context of the “semantic interoperability” work package of the QualiPSo project.

References

2. Web Service Modeling Ontology (WSMO), Submission request to W3C, June 2005
3. Semantic Web Service Framework (SWSF), W3C Member Submission July 2005
8. Evren Sirin, OWL-S API, Maryland Information and Network Dynamics Lab Semantic Web Agents Project
Semantic Web Service Discovery and Selection:
a Test Bed Scenario

Alessio Carenini², Dario Cerizza², Marco Comerio¹, Emanuele Della Valle², Flavio De Paoli³, Andrea Maurino¹, Matteo Palmonari¹, Matteo Sassi² and Andrea Turati²

¹University of Milano - Bicocca, viale Sarca 336, 20126 Milano, Italy
{comerio,depaoli,maurino,palmonari}@disco.unimib.it
²CEFRIEL - Politecnico di Milano, via Fucini 2, 20133 Milano, Italy
{carenini,cerizza,dellavalle,sassi,turati}@cefriel.it

Abstract. The Semantic Web Service Challenge is one of the major initiative dedicated to work on Semantic Web Service (SWS) discovery and selection. It represents an effective manner for evaluating the functionality of SWS technologies. In this paper, we provide a new SWS-Challenge scenario proposal with new interesting problems to be solved on the basis of an analysis of a real world shipment scenario in the logistic operators domain. In the discussion a number of aspects concerning the discovery and selection processes are emphasized. In particular, we focus on the problem of considering the heterogeneity between the provider and the requester perspectives, and on the differences between functional and non functional specifications both on the requester and provider side.

Key words: Semantic Web Services Discovery, Semantic Web Services Selection, Non-functional properties, Semantic Matching, SWS-Challenge.

1 Introduction

Currently, Web services (WSs) technology enables the implementation of the Service Oriented Computing paradigm to develop Web processes that are accessible within and across organizational boundaries [6]. Web services provide for the development and execution of business processes that are distributed over the network and available via standard interfaces and protocols. Nevertheless, there is a growing consensus that pure functional descriptions of WSs are inadequate to develop valuable processes due to the high degree of heterogeneity, autonomy, and distribution of the Web [1]. Moreover, the increasing availability of Web services that offer similar functionalities with different characteristics increases the need for more sophisticated discovery processes to match user requests.

A way to address this problem is to combine Semantic Web and Web service technologies. The resulting Semantic Web service (SWS) technology supports the (semi)-automatic development of Web processes that go beyond simple interactions between organizations. The focus is on the discovery and selection activity that is requested to identify the needed Web services and supply the
business information to include them in a business process. The idea is that a number of SWSs can meet some basic requirements specified by a requester. Then, a requester needs to be supported in choosing which one of the above SWSs better fulfills a set of desiderata; the selection can be automatic, e.g. automatically proceeding to invoke the “best” service (“best” according to a set of chosen criteria), or left to the discretion of a human requester.

Today, there is not yet a clear agreement on the definition of the terms discovery and selection. In this paper we call discovery the activity of locating a machine-processable description of a SWS that meets certain functional criteria [8]; and selection the activity of evaluating the discovered services to identify the ones that better fulfill a set of non-functional properties (NFPs) requested by the actual user [4], whatever the final invocation step is automatized or user-driven. In our vision, the selection process can be considered an enrichment of the discovery process.

A research topic that has been recently addressed concerns the support to NFP-based service selection (see [4, 7] and references therein); with respect to this problem, different models for NFP request and offer specification have been proposed. Typically, models based on attribute-value specifications are used [9]. An attribute identifies the involved NFP and the value quantifies it. This approach is effective and intuitive when simple specifications are represented (i.e. the value is known, fixed, and not dependent on any condition). However, the attribute-value approaches have two main shortcomings. First, an explicit representation of joint offers of NFP is not natively supported; in order to represent such joint offers there is the need to use a set of axioms establishing same conditions for set of values. Second, the expressiveness of the attribute-value approach is quite poor. NFP requests and offers make use of constraint expressions involving constraint operators that are not usually considered. In order to address such kinds of requests, new language constructs for NFP requests should be introduced. In the literature NFP models overcoming some limits of the attribute-value approach have been proposed (e.g. [5, 3]). Some of the authors of this paper proposed a NFP model in order to handle some of the above shortcomings of the attribute-value approach [2, 8]; this model is based on the concept of Policy (i.e., a joint offer or request of NFPs) and on the explicit representation of offers and requests through constraints.

The Semantic Web Service Challenge is one of the major initiative dedicated to SWS discovery and selection. The SWS-Challenge provides test bed scenarios and consists of a set of problems to be solved by the participants. The SWS-Challenge is an effective manner for evaluating the functionality of SWS technologies. One of the scenarios of the SWS-Challenge is about the problem of discovering a shipment service able to satisfy a set of user requirements (e.g., the pick-up location and time). This scenario points out the necessity to consider both functional and non-functional characteristics of SWSs.

By starting from that fictitious scenario, in this paper we propose a new similar scenario that is more realistic because it is based on a real domain. For this reason, we provide a detailed description of the activities and characteristics
of logistic operators in order to acquire the needed knowledge about the domain. We will focus on a number of aspects concerning the discovery process, such as the problem of considering the heterogeneity between the provider and the requester perspectives, and the differences between functional and non-functional specifications both on the requester and provider side. This new test bed scenario will introduce new interesting problems to be solved.

The paper is organized as follows. Section 2 describes the logistic operator domain, while Section 3 highlights important aspects that should be considered in modeling such domain. Then, Section 4 proposes a novel shipment discovery scenario for the SWS-Challenge. Finally, Section 5 draws conclusions.

2 The Logistic Operator Domain

In order to provide a fully featured test for semantic Web services, a structured knowledge-intensive description of a real domain is necessary. The logistic operator domain has all the features to attain a complete testing for semantic Web services: (i) well defined terminology; (ii) well defined processes; (iii) complex normative; (iv) different points of views between the service provider and the requester.

This realistic description has been realized within the Networked Peers for Business (NeP4B) project through (i) the analysis of several real-world logistic operator prices and service offer lists, (ii) several phone and face to face interviews with logistic operators and client companies and (iii) the analysis of the normative that regulates the logistic operator activities. In the analyzed domain there are different kinds of actors (logistic operator, client company and broker) each one with his features and objectives. The primary objective of a logistic operator is to provide logistic support to third parties. Client companies, instead, need a logistic operator for their transportation, in order to fulfill a single or a continuous need, i.e. an organizational emergency or the outsourcing of the logistic coordination. A broker is a company with the intent of collect the service offers of several small logistic operators, in order to offer a more complete service. For a client company there’s no real difference between a broker or a single logistic operator.

The logistic support offered by logistic operators can concern freight transport and warehousing which are complex and structured services. In freight transport service, the logistic operator makes a delivery of a freight, in an agreed location, within some time constraints. In warehousing service, the logistic operator stores a certain amount of goods for some time in one of its warehouse. A logistic operator can provide only freight transportation service or only warehousing service or both the services, even in combination.

Freight transport and warehousing regard different kinds of goods. Each kind must be treated in different ways and needs a specific means of transportation i.e. classes of trucks. Mainly, the categories of goods are: (i) ordinary; (ii) perishable; (iii) dangerous and (iv) cumbersome. Ordinary goods do not need a

\[\text{1}\] \(\text{http://www.dbgroup.unimo.it/nep4b/en/index.htm}\)
specific way or particular attention to be treated. They can be transported by ordinary trucks, with no particular features. An ordinary load can be transported aggregated with other kinds of ordinary goods (groupage). Perishable goods, instead, need a particular temperature that must be maintained for all the transport duration. Perishable goods can be fresh, frozen or deep-frozen and need to preserve different temperature during the transportation. This means that not every perishable goods can be transported together with other ones. Dangerous goods, need a particular planning of the freight and particular treatment. Cumbersome goods are particular loads characterized by big dimensions and weight. The transport of this kind of good is regularized by the local rules of the road (exceptional transport).

In order to provide a logistic support service, a logistic operator must have a fleet and/or a logistic platform, i.e. a set of means of transport and/or a set of warehouses in different locations. The features of every kind of truck are stated by a specific normative. For example, the Accord Transport Perissable (A.T.P.) normative describes several classes of trucks with their temperature range. Warehouses are characterized by several properties as the kind of goods that can be stored, the minimum and/or maximum dimension and weight of a load and a weekly based calendar with its opening time.

3 Discussion: Aspects to be Considered

A number of aspects need to be considered in the above described scenario. In particular in what follows we focus on the need to consider different perspectives for the providers and the requesters of the service, and on the necessity to address both functional and non-functional properties of the service.

3.1 The Requester and Provider Perspectives in the Logistic Operator Domain

In the process of service discovery one must keep in mind that there are different points of view when describing a logistic operator service offer or a client company request. In fact the logistic operator has its terminology for the description of the services that a client company may not know or understand. A logistic operator: (i) has a fleet; (ii) has a logistic platform; (iii) covers a particular geographic area; (iv) has time constraints; (v) has a base price; (vi) accepts particular payment methods; (vii) fulfills particular kinds of transportation; (viii) provides some additional services. The logistic operator fleet identifies the kinds of goods that a logistic operator can treat since specific normative states particular procedures and freight truck for every kind of goods. Other features such as carrying capacity of the trucks in the fleet may attest the ability of a logistic operator to fulfill a client company request. The logistic platform identifies the kinds of goods that a logistic operator can store, the minimum and maximum load dimensions and the location of the warehouse. The covered geographic area is where a logistic operator can pick-up or deliver a load. Freight transport and
warehousing have time constraints (e.g., the minimum quantity of hours to plan the freight) and a base price (i.e., the cost for the service invocation). A freight can be pre-paid or collect, i.e. the freight bill is at sender’s or addressee’s expense. The kinds of transportation identifies the kinds of locations that a logistic operator is willing to reach as pick-up or delivery location. The first distinction is between a specific address (domicile) or a warehouse of the logistic operator as start/end location of the freight (even in combination). Other particular kinds of transportation are the delivery to fair, to supermarket, to installation site or to remote location. Each of the particular kinds of locations requires particular procedures or means of transport and not all the logistic operators provide these kinds of services. Every logistic operator can provide additional services such as freight insurance, COD (Cash On Delivery), express deliveries or deliveries with precise and mandatory delivery instant. In addition, based upon load weight and the kind of transportation, a logistic operator may choose to do a Full-Truckload (FTL) or Less-than-Truckload (LTL) transport, i.e. aggregate several loads in one freight (cheaper) or use a means of transport for only one load (faster). Usually, LTL transport is performed only in certain locations (warehouses) in relation with certain time intervals (e.g. every Monday morning, delivery in Milan, etc).

A client company may express its need in a whole different way. A request is characterized by: (i) a quantity of a particular kind of goods; (ii) a pick-up location; (iii) a delivery location; (iv) time constraints; (v) a payer; (vi) additional features. The first point identifies the load to be transported. The pick-up and delivery locations identify the address and the kind of location of the freight (warehouse, domicile, etc.). A transport has a start and an end time identifying the interval between the pick-up and the delivery of the load. In a request is even specified at who’s expense is the freight bill, i.e. the addressee or the sender. In the additional features, a requester may specify preferences related to the possibility to aggregate the load with other loads, the freight insurance, the delivery time and particular truck features.

3.2 Functional and Non Functional Properties

Typically, functional properties (FPs) of services represent the description of the service functionalities and non-functional properties (NFPs) represent the description of the service characteristics (e.g., business aspect, QoS) which are not directly related to the functionality provided [2, 8].

Nevertheless, this distinction is ambiguous and does not provide a set rule to qualify a property as FP or NFP. From our point of view this is a consequence of the fact that functional or non-functional is not an intrinsic qualification of a property, but it depends on the application domain and context. For example, the service location could be classified as a FP for a logistic service and as a NFP for a payment service. Moreover, from the requester perspective, the classification of requested properties with respect to FP/NFP is of little interest and counterintuitive. The requested properties represent the user preferences and could be mandatory and optional.
In this paper, we distinguish between the provider and the requester perspectives and we propose the following classification of properties. From the requester perspective, we considered hard and soft constraints to distinguish between the properties that are specified as mandatory or optional in a service request. From the provider perspective, we consider FPs those properties of a service that strictly characterized the offered functionality (e.g., service location for a shipment service) and NFPs those properties of a service that do not affect or affect only marginally the offered functionality (e.g., service location for a payment service).

The above consideration makes the problem of matching between requested and offered properties more difficult to handle. A possible approach consists in mapping FPs and NFPs with hard and soft constraints respectively. Note that, even if NFPs are considered as soft constraints, they could be quite relevant in order to select a service that better matches the user request. The selection should occur among a set of services that satisfy the specified hard constraints. In practice, the role of hard constraints is to define a boolean match that filters eligible services. On the contrary, matching between soft constraints should be considered not crispy and degrees of satisfaction should be evaluated (i.e., soft constraints can be fully, partially or not satisfied). In order to support non-boolean matching a number of characteristics should be considered:

- **Support for sophisticated descriptions**: Requests, and also offers, should refer to a full set of constraint operators, including “≤”, “≥” and range. Moreover, relevance attributes should specify the importance that requesters give to each requested property. The use of relevance attributes and operators in constraint expressions (e.g., the cost of the service must be $\leq 3$ Euro) enhance the expressivity of the descriptions to support non-boolean matching.

- **Support for offer clustering**: A Web service can be offered with different levels of NFPs. For example, a SMS service offered by a telephone company can be characterized by different policies: one offering a price of 0.10 Euro for SMS and a maximum of 100 SMS per day; another offering a price of 0.15 Euro and no restriction on the number of SMS per day. Moreover policies can be associated with conditions to state that their applicability depends on the requester’s profile or context; e.g. a condition may grant a discount on shipment services to registered customers or for multiple shipment requests.

- **Support for static and dynamic properties description**: Static NFPs specify properties of services by fixed values that can be included in service descriptions at publishing time. Dynamic NFPs describe properties of services that are strictly related to the context of execution and cannot be stated at publishing time. Moreover, dynamic NFPs require service calls to fix the actual values. For example, the actual price for shipment depends on the shipped object’s weight, therefore a call is necessary to fix it.
4 New SWS-Challenge Proposal

The current discovery scenario of the SWS-Challenge is based on shipment services offered by generic logistic operators. These services are defined mainly in terms of the time required to delivery the goods, the covered geographical area and the price. In sections 2 and 3 we have described several features that must be taken into account when modeling the logistic operator domain. In this section we propose a new shipment discovery scenario by defining four logistic operator web services and two user goals. As our logistic operator domain description offers much more details with respect to the one given by the current scenario, we decided to use different names for the services. The reason is that we do not merely add more attributes, but we redefine some concepts that were already present in the previous shipment service descriptions.

4.1 Logistic operators

“Liteworld” is an operator specialized in transcontinental deliveries of small ordinary goods. Its features are listed below.

- Price (flat fee/each kg): Europe(41/6.75), Asia(47.5/7.15), North America(26.25,4.15), Rates for South America like North America, Rates for Oceania like Asia
- Performs LTL delivery
- Maximum weight: 5 kg
- Transports to/from domicile
- Covered geographic area: Africa, Europe, North America, Asia
- Fleet: 20 trucks for ordinary goods
- Freight insurance: refund for loss
- If number of requested shipments $\leq 5$:
  - Base price: 50 Euro
  - Minimum hours to delivery: 96 hours
- If number of requested shipments $> 5$:
  - Base price: 30 Euro
  - Minimum hours to delivery: 72 hours

“Hello” delivers ordinary goods and performs exceptional freight deliveries in three European countries and in Asia:

- Price on request
- Performs LTL and FTL delivery
- Transports to/from domicile
- Minimum time to deliver: 48 hours
- Covered geographic area: Italy, France, Germany, Asia
- Fleet: 10 trucks for ordinary goods, 12 trucks for exceptional transports
- Freight insurance: refund for damage and refund for loss
- Price:
  - Base price: 35 Euro
• Base price discount if number of requested shipments $\geq 10$: 20%
• Base price discount if number of requested shipments $\geq 20$: 50%

“Fresh and fast” has a fleet that allows it to deliver fresh goods (ATP IN) as well as frozen goods (ATP IR). Its two warehouses allow this operator to perform transports to and from warehouses:

– Price on request
– Performs LTL and FTL delivery
– Transports to/from domicile and to/from warehouse
– Minimum time to deliver: 24 hours
– Covered geographic area: Italy, France, Germany
– Fleet: 10 trucks for ordinary goods, 5 trucks for ATP IN goods, 5 trucks for ATP IR goods
– Warehouses: see table 1

<table>
<thead>
<tr>
<th>Location</th>
<th>W1</th>
<th>W2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accepted goods</td>
<td>ATP (IN and IR)</td>
<td>ATP (IN and IR)</td>
</tr>
<tr>
<td>Minimum weight</td>
<td>5 q</td>
<td>5 q</td>
</tr>
<tr>
<td>Minimum storage time</td>
<td>3 days</td>
<td>5 days</td>
</tr>
</tbody>
</table>

“Safe transports” is a specialized operator that performs transports of frozen and dangerous goods:

– Price on request
– Performs FTL delivery
– Transports to/from warehouse
– Minimum time to deliver: 48 hours
– Covered geographic area: Belgium, France, Spain
– Fleet: 3 trucks for dangerous (ADR) goods, 5 trucks for ATP IR goods
– Warehouses: see table 2

<table>
<thead>
<tr>
<th>Location</th>
<th>W1</th>
<th>W2</th>
<th>W3</th>
<th>W4</th>
<th>W5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accepted goods</td>
<td>ATP</td>
<td>ATP</td>
<td>ADR</td>
<td>ADR</td>
<td>ADR</td>
</tr>
<tr>
<td>Minimum weight</td>
<td>20 q</td>
<td>10 q</td>
<td>10 q</td>
<td>10 q</td>
<td>20 q</td>
</tr>
<tr>
<td>Minimum storage time</td>
<td>3 days</td>
<td>3 days</td>
<td>3 days</td>
<td>3 days</td>
<td>3 days</td>
</tr>
</tbody>
</table>
4.2 Goals

A first goal requires the delivery of fruit ice cream which requires an ATP IN truck. The goods will be brought by the customer to a warehouse that has to be located in Paris and have to be delivered to a warehouse in Cannes. This goal tests both the fleet of the operator and the features of its warehouses.

- Freight type: Fruit ice cream
- Weight: 10 q
- Source: Paris, France
- Destination: Cannes, France
- Other requirements: Transport from warehouse to warehouse

This goal will be satisfied by the “Fresh and fast” operator. One of the warehouses of “Safe transports” has a minimum weight for incoming freight that is higher than the weight of the ice cream, while neither the fleet of the “Hello” operator nor the one of the “Liteworld” operator can deliver perishable goods.

We can now introduce non functional properties while expressing a second goal, which requires the transport of piles of documents from Singapore to Turin. This goal states preferences about the price, the maximum time to deliver and the freight insurance. This means that if two operators can fulfill the functional requirements, the one that better satisfies the non functional requirements have to be chosen.

- Freight type: Documents
- Weight: 3 kg
- Source: Singapore
- Destination: Turin, Italy
- Base price < 30 Euro (preferred)
- Hours to delivery ≤ 72 hours (preferred)
- Freight insurance: refund for damage and refund for loss (preferred)
- Collaboration period: from 2008-03-03 to 2008-05-26
- Transport frequency: weekly

The request expressed by this goal is fulfilled by both “Liteworld” and “Hello”. “Safe transports” and “Fresh and fast” do not match the requests, as the former performs only FTL goods delivery and the latter does not cover the source area. As “Hello” better satisfies the optional requests related to Freight insurance, Base price and Minimum hours to delivery, it should be preferred over “Liteworld”.

5 Conclusion

The Semantic Web Service Challenge is one of the major initiative dedicated to SWS discovery and selection. The SWS-Challenge provides test bed scenarios and consists of a set of problems to be solved by the participants. In this paper we have proposed a new shipment discovery scenario based on an analysis of
the logistic operator domain realized within the Networked Peers for Business (NeP4B) project. The new scenario considers the necessity of: (i) addressing the heterogeneity between the provider and the requester perspectives and (ii) specifying functional and non-functional properties both on the requester and provider side. This new scenario can be considered an effective manner for evaluating the possibility to use SWS technologies for generating requested and offered property descriptions useful for the discovery and selection activities.

**Acknowledgements**

The work presented in this paper has been partially supported by the European IST project n. 27347 SEEMP - Single European Employment Market-Place and the Italian FIRB project RBNE05XYPW NeP4B - Networked Peers for Business.

**References**

Abstract. In this paper we show how to apply a tableau-based software composition technique to automatically generate the mediator’s service logic. This uses an LTL planning (or configuration) algorithm originally embedded in the ABC and in the ETI platforms. The algorithm works on the basis of the existing jABC library of available services (SIB library) and of an enhanced description of their semantics given in terms of a taxonomic classification of their behaviour (modules) and abstract interfaces/messages (types).

1 The SWS Challenge Mediator

The ongoing Semantic Web Service Challenge [19] proposes a number of increasingly complex scenarios for workflow-based service mediation and service discovery. We use here the technology presented in [10] to synthesise a process that realizes the communication layer for the Challenge’s initial mediation scenario.

In this scenario, a customer (technically, a client) initiates a Purchase Order Request specified by a special message format (RosettaNet PIP3A4) and waits for a corresponding Purchase Order Confirmation according to the same RosettaNet standard. The seller however does not support this standard. Its backend system or server awaits an order in a proprietary message format and provides appropriate Web Services to serve the request in the proprietary format. As client and server here speak different languages, there is a need for a mediation layer that adapts both the data formats and also the granularity.

Of course we can easily define the concrete process within our jABC modelling framework, as we have shown in the past [11, 6, 7].

To provide a more flexible solution framework, especially to accommodate later declarative specification changes on the backend side or on the data flow, we synthesize the whole mediator using the synthesis technology introduced in [10]. We proceed here exactly along the lines already presented in that paper.

In the following, we show in Sect. 2 how to use the SLTL synthesis methodology to generate the mediator workflow based on a knowledge base that expresses the semantics
of the concrete types from the SWS mediator scenario, then in Sect. 3 we add a more business-level-like abstraction to the knowledge base, and in Sect. 4 we show how this leads to a looser solution, and how this solution can be stepwisely refined towards the first solution by adding business-level knowledge to the problem definition, in a declarative way. Sect. 5 describes how to work with the synthesis tool. Finally, Sect. 6 discusses related work and Sect. 7 draws some conclusions and sketches ongoing work.

2 The Concrete Mediator Workflow

2.1 Abstract Semantics: Taxonomies for Modules and Types

Table 1 shows the modules identified within the system. They represent at the semantic level the collection of basic services available for the mediator. In order to produce a running solution as demonstrated in Stanford in November they are then bound (grounded) to the concrete SIBs that in the jABC constitute the running services. How


<table>
<thead>
<tr>
<th>module name</th>
<th>input type</th>
<th>output type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mediator</td>
<td>PurOrderReq</td>
<td>PurOrderReq</td>
<td>Maps RosettaNet messages to the backend</td>
</tr>
<tr>
<td>startService</td>
<td>{true}</td>
<td></td>
<td>Receives a purchase order request message</td>
</tr>
<tr>
<td>obtCustomerID</td>
<td>PurOrderReq</td>
<td>SearchString</td>
<td>Obtains a customer search string from the req. message</td>
</tr>
<tr>
<td>createOrderUCID</td>
<td>CustomerObject</td>
<td>CustomerID</td>
<td>Gets the customer id out of the customer object</td>
</tr>
<tr>
<td>buildTuple</td>
<td>OrderID</td>
<td>Tuple</td>
<td>Builds a tuple from the orderID and the POR</td>
</tr>
<tr>
<td>sendLineItem</td>
<td>Tuple</td>
<td>LineItem</td>
<td>Gets a LineItem incl. orderID, articleID and quantity</td>
</tr>
<tr>
<td>closeOrderMed</td>
<td>SubmConfObj</td>
<td>OrderID</td>
<td>Closes an order on the mediator side</td>
</tr>
<tr>
<td>confirmLIOperation</td>
<td>OrderConfObj</td>
<td></td>
<td>Receives a conf. or ref. of a LineItem and sends a conf.</td>
</tr>
<tr>
<td>Moon</td>
<td>SearchString</td>
<td>CustomerObject</td>
<td>The backend system</td>
</tr>
<tr>
<td>searchCustomer</td>
<td>OrderID</td>
<td></td>
<td>Gets a customer object from the backend database</td>
</tr>
<tr>
<td>createOrder</td>
<td>CustomerID</td>
<td>OrderID</td>
<td>Creates an order</td>
</tr>
<tr>
<td>addLineItem</td>
<td>LineItem</td>
<td>SubmConfObj</td>
<td>Submits a line item to the backend database</td>
</tr>
<tr>
<td>closeOrderMoon</td>
<td>OrderID</td>
<td>Timeout</td>
<td>Closes an order on the backend side</td>
</tr>
<tr>
<td>confRefLineItem</td>
<td>Timeout</td>
<td>orderConfObj</td>
<td>Sends a conf. or ref. of a prev. subm. LineItem</td>
</tr>
</tbody>
</table>

Table 1. The SWS mediation Modules

This happens is sketched in [17].

This information about the single modules is complemented by simple ontologies that express in terms of is-a and has-a relations properties over the types and the modules of the scenario. We call these relations Taxonomies. The taxonomies regarding the mediation scenario are shown in Fig. 1 (Type Taxonomy) and Fig. 2 (Module Taxonomy).

This information is expressed in a Prolog-like fashion in a concrete knowledge base which feeds then the synthesys algorithm.

2.2 The Concrete Knowledge Base

The synthesis tool takes as input a textfile with the definitions of the taxonomies (module and type taxonomy), the module descriptions, and some documentation. The first line of the file declares a name for the knowledge base:

$program(sws_challenge).

The file contains statements (one per line) of facts in the following three forms:

- `tax(type, output, customerObject).`
- `tax(module, mediator, sendLineItem).`
- `module(searchCustomer, searchString, customerObject).`

The two first statements show how to specify the type and module taxonomy:

- The first line declares `customerObject` as a subtype of the `output` type.
- The second line declares module `sendLineItem` to be a `mediator` module.
The third statement form is used to specify the relation between input and output types for particular modules. It describes the module definition as already presented in Table 1: the searchCustomer module takes a searchString as input type and produces a customerObject output type.

This way it is possible to concisely represent the taxonomies of Fig. 1 and 2 as well as the module description of Table 1 in one single file.

### 2.3 Semantic Linear-time Temporal Logic

The loose specification language supported by the synthesis is the Semantic Linear-time Temporal Logic (SLTL)[14], a temporal (modal) logic comprising the taxonomic specifications of types and activities. This lifts the classical treatment of types and activities in terms of actions and propositions to a semantical level in a way typical today in the context of the semantic Web.

**Definition 1 (SLTL).**

The syntax of Semantic Linear-time Temporal Logic (SLTL) is given in BNF format by:

\[
\Phi ::= \text{type}(t_c) | \neg\Phi | (\Phi \land \Phi) | <a_c> \Phi | G(\Phi) | (\Phi U \Phi)
\]

where \(t_c\) and \(a_c\) represent type and activity constraints, respectively, formulated as taxonomy expressions.

SLTL formulas are interpreted over the set of all legal coordination sequences, i.e. alternating type correct sequences of types and activities\(^1\), which start and end with types. The semantics of SLTL formulas is now intuitively defined as follows:\(^2\):

- \(\text{type}(t_c)\) is satisfied by every coordination sequence whose first element (a type) satisfies the type constraint \(t_c\).
- Negation \(\neg\) and conjunction \(\land\) are interpreted in the usual fashion.
- **Next-time** operator \(<>\):

  \(<a_c> \Phi\) is satisfied by coordination sequences whose second element (the first activity) satisfies \(a_c\) and whose *continuation*\(^3\) satisfies \(\Phi\). In particular, \(<tt> \Phi\) is satisfied by every coordination sequence whose continuation satisfies \(\Phi\).

- **Generally** operator \(G\):

  \(G(\Phi)\) requires that \(\Phi\) is satisfied for every suffix\(^4\) satisfies \(\Phi\).

\(^1\) During the description of the semantics, types and activities will be called *elements* of the orchestration sequence.

\(^2\) A formal definition of the semantics can be found online.

\(^3\) This continuation is simply the coordination sequence starting from the third element.

\(^4\) According to the difference between activity and type components, a suffix of a coordination sequence is any subsequence which arises from deleting the first 2n elements (n any natural number).
Until operator $U$:

$(\Phi U \Psi)$ expresses that the property $\Phi$ holds at all type elements of the sequence, until a position is reached where the corresponding continuation satisfies the property $\Psi$. Note that $\Phi U \Psi$ guarantees that the property $\Psi$ holds eventually (strong until).

The definitions of continuation and suffix may seem complicated at first. However, thinking in terms of path representations clarifies the situation: a subpath always starts with a node (type) again. Users should not worry about these details: they may simply think in terms of pure activity compositions and not care about the types, unless they explicitly want to specify type constraints.

The online introduction of derived operators supports a modular and intuitive formulation of complex properties.

### 2.4 Declarative LTL Specification for the Concrete Mediator

For the mediator, we look for a workflow (a service coordination) that satisfies the following requirement:

*The mediator service should produce a Purchase Order Confirmation.*

The corresponding formal specification formulated in SLTL is simple: we need to start the service (module $\text{startService}$) and reach the result $\text{PurOrderCon}$ (a type). We
may simply write: \((\text{startService} < \text{PurOrderCon})\) where the symbol \(<\) denotes a derived operator meaning \textit{before} or \textit{preceeds} and is defined as

\[
f_1 < f_2 =_{df} F(f_1 \land F(f_2))
\]

The \(\text{jABC}\) process model shown in Fig. 3(a) resembles very closely the expected required solution.

If we adopt the very fine granular model of the types shown in Table 1, a natural choice given the SWS Challenge problem description, this is in fact the only solution.

In this setting, we use abstract type names in the taxonomy to model de facto almost the concrete operational semantics: we distinguish for instance an \texttt{OrderID} from an \texttt{OrderConfObject}, modelling the described application domain at the concrete level of datatypes and objects - a direct rendering of what happens at the XML level, or for programs in the memory and in the heap. This is however already a technical view, and it corresponds to lifting the concrete, programming-level granularity of data to the semantic level: the resulting ontology is as concrete as the underlying program.

This is however not the intention of Service Orientation, nor of the semantic web: the idea there is to decouple the business-level view (captured at the semantic level) from the technical view of a specific implementation, in order to allow a coarser description of business-level workflows and processes that then must be concretized and grounded to a running implementation. In the following we show how this can be done, also including automatic synthesis.

3 Abstract Semantics: Using Abstraction and Constraints

For a specifier and definer of the business domain it is much more realistic to say that the modules concerned with orders work on an \texttt{Order} type, which is a business-level abstraction for order-related objects and records, and to leave the distinctions to a problem-specific refinement of the desired solutions via constraints added at need.

For the abstract semantics we work on the taxonomies. The taxonomy design and module specification decides here the balance between concreteness and flexibility (looseness). In this specific case, we change the definition of the modules that deal with orders as shown in Tab. 2: they now operate on the abstract \texttt{Order} type. We can be as concrete, or as abstract and generic as we wish, and choose the suitable description level driven by the semantics or application domain modelling. This abstraction determines how much flexibility we build in into our solutions. At the one extreme we can have very specific types, as fine granular as a description in terms of structural operational semantics [12]. In this case, solutions are type-determined, and basically render the concrete labelled transition system underlying the manually programmed solution as in Fig. 3(a). At the other extreme one could also model the process structure solely by means of temporal constraints. However, most flexible is a hybrid approach which combines loose taxonomies and module descriptions with temporal constraints in order to arrive at an adequate specification formalism.

No matter the choice, the algorithm covers the whole spectrum, leaving it free to the application domain designer to determine where to be precise and where to be loose, leaving space for exploring alternatives and tradeoffs.
4 A Loose Solution, and its Declarative Refinement

4.1 The base case

If we now solve the planning problem with the modified module description and the original goal, we obtain a much shorter solution, shown in Fig. 3(b). This is due to the fact that these module specifications now refer to the abstract type Order. As a consequence, closeOrderMoon is a suitable direct successor of createOrder. This solution corresponds to a degenerate workflow where an empty order is sent.

4.2 Refinement1: Nonempty Orders

Since in the normal case orders contain items, the business expert needs to be more precise in the specification of the solution, adding knowledge by means of SLTL constraints. If one just knows that the items are referred to via the LineItem type, one may simply refine the goal as follows:

\[(\text{startService} < \text{LineItem} < \text{PurOrderCon})\]

This way, we have added as additional intermediate goal the use of a LineItem type. Accordingly, at least one of the modules \{addLineItem, sendLineItem\} must appear in the required minimal workflow. We see the result in Fig. 4(a): this solution coincides with the previous one till the createOrder module, then the type mediator buildTuple is added, after which sendLineItem satisfies the intermediate goal. The remaining constraint at that point is simply the reaching of the final type PurOrderCon, which is done by generating the sequence CloseOrderMediator followed by CloseOrder.

This solution however corresponds only to the first Web service realizing the mediator. There is in fact a subsequent second service that realizes the confirmation part of the mediator.

4.3 Refinement2: Confirmed Nonempty Orders

To obtain this part as well, we have to additionally specify that we need to see a confirmation, e.g. as confRefLineItem module:
Fig. 4. (a) Adding a LineItem: the new solution and (b) Adding a Confirmation: the complete loose solution

\[(\text{startService} < \text{LineItem} < \text{confRefLineItem} < \text{PurOrderCon})\]

This generates the solution of Fig. 4(b), which includes also the rest of the sequence shown in Fig. 3(a).

5 How to work with the Synthesis Tool

The synthesis tool takes as input the text file containing the knowledge base: the module and type taxonomy, the module descriptions, and some documentation for the integrated hypertext system. It is steered from the ABC GUI. There, users can input the SLTL formulas that describe the goal and can ask for different kinds of solutions. The tool produces a graphical visualization of the satisfying plans (module compositions), which can be executed, if the corresponding module implementations are already available, or they can be exported for later use.

The knowledge basis implicitly describes the set of all legal executions. We call it configuration universe, and it contains all the compatible module compositions with respect to the given taxonomies and to the given collection of modules. Fig. 5 shows the configuration universe that emerges when simply taking the atomic, concrete input/output types.
5.1 Specifying Solution Types

Users never see the configuration universe. They have a number of simple options to state which kind of solutions they would like to have displayed.

- **minimal** solutions denotes plans that achieve the goal without repetition of configurations. In particular, this excludes cycles.
- **shortest** solutions returns the set of all minimal plans that are also shortest, measured in number of occurring steps.
- **one shortest** solution returns the first shortest plan satisfying the specification.
- **all** solutions returns all the satisfying solutions, which includes also cyclic ones.

Minimal plans generated for our working example are shown in Fig. 6. Since these plan descriptions are directed acyclic graphs, it is rather simple to select and execute one plan.

The typical user interaction foresees a successive refinement of the declarative specification by starting with an initial, intuitive specification, and asking typically for shortest or minimal solutions, and using the graphical output for inspection and refinement.

This is exactly what we did in Sect. 3, where we used abstract types to enlarge the solution space and then tightened successively the LTL specification by adding salient characteristics that yield a good declarative characterization of the desired solutions.
6 Related Approaches

Our approach was introduced 1993 ins [15, 4] and applied in [16, 9] and [18] to synthesize Statecharts, CAD design processes, and heterogeneous verification algorithm for concurrent systems, respectively. The idea of LTL guided process composition has later been taken up by others: Bacchus and Kabanza [1]extensively discuss their technique that implements a first order extension of LTL, Mandell and McIlraith use LTL in the context of BPEL compositions [8], and Falcarin et al. [20] use LTL as a starting point for their compositions, transforming single LTL formulas to finite state automata, then composing them to a global specification, and finally finding the correct shortest solutions as the acyclic accepting paths in that automaton.
Concerning the relation with planning, the state variables in an LTL formula are directly fluents: their value changes from state to state along the process, and the formulas describe mutual dependencies naturally and compactly. In this sense, there is a close kinship between the temporal logic mentality and event calculus [13] or logics for timing diagrams [3]: all three describe what is true at what time, associating the evolution of time with a succession of states, and offering a well chosen set of operators to express dependencies between temporal variables along possible paths within models. The fundamental advantages of LTL guided synthesis over planning are the following:

- the guidance it allows is process driven and not state driven. Therefore the control it offers can in general depend on the entire history of predecessors, and not only on the current state. This is extremely efficient in focusing the search, resulting in small memory usage and quick execution.
- it is decoupled from the (internal) state of a solver/planner: the search control information relates exclusively to properties of the domain knowledge, not on any information on the internal state of an algorithm, which is often the case for planning techniques in order to capture and encode the relevant history aspects (what is enabled, what is true, etc.) that govern the correct chaining of transitions, i.e. the temporal/causal/precedence aspects. In contrast, a user of our technique does not need to know anything about the algorithm underlying the solver/planner.

7 Conclusions

We have applied the automatic tool composition feature of the ABC/ETI platform as a synthesis tool for the mediator. Our LTL-based synthesis approach is not restricted to compute one solution, but it may compute all (shortest/minimal) solutions, with the intent to provide maximum insight into the potential design space.

In future we plan to investigate various forms of synthesis approaches in order to compare their application profiles. In particular, we are interested in comparing game-based methods which work via synthesis of winning strategies with the described tableau-based methods, that construct a linear model as a result of proof construction. We also plan to enhance the user-friendliness in terms of graphical support for the declarative specifications, for example by means of the Formula Builder [5] and by the use of patterns [2].

References

Abductive Synthesis of the Mediator Scenario with jABC and GEM

Christian Kubczak1 Tiziana Margaria2 Matthias Kaiser3 Jens Lemcke4 Björn Knuth2

1 Chair of Software Engineering, Dortmund Univ. of Technology, 44227 Dortmund, Germany, christian.kubczak@cs.uni-dortmund.de
2 Chair of Service and Software Engineering, Universität Potsdam, 14482 Potsdam, Germany, margaria@cs.uni-potsdam.de
3 SAP Res. Center Palo Alto, SAP Labs LCC, 3410 Hillview Ave, Palo Alto, CA 94304, USA, matthias.kaiser@sap.com
4 SAP Research, CEC Karlsruhe, SAP AG, Vincenz-Prießnitz-Str. 1, 76131 Karlsruhe, Germany, jens.lemcke@sap.com

Abstract. We reuse here the framework, the setting, and the semantic modelling for the automated synthesis of the SWS Challenge Mediator presented in the companion paper [5], and show how to exchange the synthesis paradigm from a linear-time-logic proof based algorithm to a runtime, abductive synthesis implemented in Prolog, that responds directly to situational changes. Both solutions take advantage of a policy-oriented enterprise management approach.

1 Motivation

The Semantic Web Service Challenge Mediation Scenario has been solved in [5] by synthesis of its business logic within jABC [1]. There, we use semantic annotations for data and services expressed via type and activity taxonomies, and we derive automatically several versions of the business logic by means of a proof algorithm for the (semantic) linear time logic SLTL. We showed thus that, given high-level, declarative knowledge about the goals to be achieved by the mediator, and abstract information on the available services, their purpose, and their compatibility conditions, we could automatically produce the set of processes that solve the goal and satisfy the constraints. This way, we produced a set of solutions that are correct by construction, are at the same time executable service orchestrations, and are amenable to investigation for compliance wrt. additional criteria, since these orchestrations can be directly analyzed with jABC’s mechanisms, e.g. by model checking.

Such an approach has the typical advantages of design-time solutions: they can be saved, compiled for specific platforms, thus living outside the environment where they are computed, and are adequate for documentation, auditing, and massive reusal.

In cases where the environment is highly dynamic it may however be desirable to have a more reactive generation of solutions. Typically this leads to on-the-fly approaches, where the (piecewise) creation of the solution and its (possibly piecewise) execution happen simultaneously. SAP’s Goal-oriented Enterprise Management (GEM)
A typical advantage of such approaches is their situation awareness: changes in the environment are directly taken into account while building the solution. The typical drawback is that the existence of a solution cannot be guaranteed, and if the solution space empties, expensive backtracking and rollbacks are needed. Also, justification and further analysis can only be carried out by a-posteriori validation.

An ideal environment should therefore offer both paradigms and enable the user to choose from case to case. This is what we want to achieve with the jABC. To this aim, it is important that both synthesis approaches foot on the same semantic descriptions, use the same information in roughly the same way, and can share the same grounding.

In this paper, we show how both approaches are instances of business knowledge-driven process synthesis [6, 3], and how we can accommodate both in the jABC environment in a modular way: we exchange the synthesis method with SAP’s GEM approach to automatically solve the same mediation task using the same domain modelling and service grounding.
1.1 Agent-based Mediator Synthesis.

One first step towards the solution is to implement the idea behind this on-the-fly concept within a working prototype. The basic idea is to construct a set of agents that interact with the company’s system and find parts of workflows as solutions to goals in a dynamic environment. The concept behind the behavior of an agent is presented in [3] and illustrated here in Fig. 1. In this reference model, a GEM agent refers to situations and goals in order to identify suitable subgoals that, in the given situation, can be reduced via business rules resulting in a fragment of a plan that can be enacted.

We implemented the GEM reference agent as a jABC orchestration: there is a working jABC process, implemented as a service, for each single agent component, shown later in fig. 4, and the jABC orchestration of Fig. 1 corresponds to the workflow of the reference description. Each agent component Goal Hypothesizer, Guide, Explainer, Analyzer, Performer is implemented as a Prolog algorithm that is accessed in the jABC as a jETI service hosted on a remote machine. External resources are the collections of BusinessExecutionPolicies, AgentScopePatterns, AgentPurposeRules, that feed the single agents with knowledge, and the collection of BusinessObjects on which the agents operates. They are also hosted elsewhere and accessed as services. Additionally, Explanations that justify the decisions taken can be consulted.

In order to understand how this works in practice, we need first to introduce abductive reasoning, the event calculus, and to explain how the GEM synthesis process is organized.

In the following, Section 2 sketches abductive reasoning and the event calculus, Section 3 describes the Goal-oriented Enterprise Management system (GEM), and Section 4 shows how to use GEM to solve the first Mediation scenario. Finally a conclusion is drawn and the ongoing tasks are presented in section 5.

2 Abductive reasoning

In contrast to the traditional composition, Our general approach towards synthesizing business processes is based on the notion of abductive reasoning. We synthesize a goal-driven process on the basis of available services represented as descriptions of pre- and postconditions (or conditions and effects), as well as policies which constrain which goals are admissible (main goal and subgoals) and how to achieve them. We now take a brief look at the nature of abductive reasoning, how it is realized in ways of programming and then describe the process synthesis by ways of a planner implementing a form of abductive process synthesis.

Abductive reasoning is also called reasoning to the best explanation. It is the kind of reasoning used to hypothesize causes on the basis of observed results. Thus one of its main applications is in diagnosis, where symptoms are observed and their causes are abduced = hypothesized. A simple way to illustrate abductive reasoning in a more logical way is to imagine the abduction step as modus ponens in a backward manner. In modus ponens, given $P \rightarrow q$, once we observe $P$, we can deduce $Q$. In abduction, given $P \rightarrow q$, once we observe $Q$ we infer that $P$ is an admissible cause of $Q$. 
The relation between deduction, abduction, and induction (another form of reasoning forming rules from observations) has been studied extensively by Peirce. The only sound kind of reasoning is deduction. However, the only way to obtain new knowledge is by 1) using abduction and induction and 2) then prove the validity of newly obtained facts or rules within a previously formed theory (where refinement of this theory is also an option in case of conflicts with new facts/rules).

A whole area in logic programming concerns the use of abduction as form of reasoning to enhance given theories by new, yet not integrated information. For the synthesis of business processes we employ the general idea of abductive logic programming.

2.1 Abductive logic programming.

Abductive logic programming is an extension of logic programming which is characterized by the possibility that predicates which are not completely defined can be used to explain other predicates. I.e. the explaining (or abducible) predicates are derived by abduction. Abductive logic programs typically consist of:

- a logical program, consisting of facts and rules which can be used in the common deductive way,
- a set of abducible predicates which are incompletely defined (think of them as if parts in if-then rules)
- a set of integrity constraints which determine certain properties the program must comply with.

We now apply these notions in our domain of business process synthesis to see the connection.

We have a description of the world in form of an ordinary logic program. Situations are described by facts holding at a certain time. Services can be seen like (very complex) if-then rules with the preconditions being abducible predicates. Likewise, goals can be seen as abducible predicates which may explain other goals. Policies can be seen as constraints whose violations would lead to non-admissible programs.

Important to know is that in order to achieve a goal, i.e., complete the abducible predicate, a goal often has to be decomposed into a number of subgoals which can be achieved - normally in a certain order - so that one subgoal can be explained on the basis of the achievement of another.

If we have a business goal that we want to be achieved with a business process, then what we expect is a process that achieves the goal by execution of a sequence of services that transform an initial situation into a situation in which the goal is consistent. Such a synthesis of a process can be realized by a planning algorithm.

The algorithm underlying our planning strategy is derived from the well-known event calculus, proposed in [4] and extended by many for various purposes.

2.2 Event Calculus in Short.

Generally, the event calculus facilitates the representation and reasoning about dynamic entities involving actions and their effects using (a subset of) first order logic. This
makes it a promising approach for planning. The language of the event calculus allows
for the representation of time independently of any of events occurring.

The most common representation of event calculus can be briefly described as fol-

1. a set of time points which can be denoted by integers,
2. a set of properties, called fluents, which can vary over time, i.e., from one time
   point to another, and
3. a set of event types.

The event calculus is based on the notion of fluents which can be manipulated by
actions in time. Fluents are usually functions which are specified by the predicate holds
to be in effect at a certain time point. Fluents can be changed by actions. A fluent
is initiated, i.e., made to hold by means of an initialize, and can be terminated by a
terminate action. If we ‘count’ actions at time steps we can say whether a certain fluent
is true or false at a given time point. Translated into our domain this means that we
can regard services as actions, while looking at preconditions and effects as fluents.

The event calculus takes a number of domain-independent axioms as a foundational
theory, which is a logic program. It is enriched by domain-dependent information about
entities like constituents of situations, service preconditions and effects, and service
descriptions that state which fluents hold before and after the execution of a service.
Additionally, constraints in terms of policies can be added to “control” which changes
can legitimately happen at certain time points.

It is now straightforward to realize a planner over goals, services with their precon-
ditions and effects in compliance with formalized policies.

The planner starts the plan generation when a goal is supplied. It tries to find a
service which initiates those fluents that are described in the goal. If such a service is
found, it will be checked whether the preconditions of that service, which are expressed
as fluents, hold. If not, another service must be found whose execution will ensure that
the fluents hold and so forth. The whole planning process can be much more complex,
since more than one service might be required to satisfy goals and subgoals by initiating
required fluents. Note that ordering of services is ensured through time points that are
attached to service invocations.

The plan is complete if preconditions for a service are actually contained in the
description of the initial situation. This way, trough abduction we arrive at a plan that
can now be executed to transform the initial situation into a situation in which the
supplied goal can be achieved. The whole plan is represented in a declarative form and
it can be proven to be correct given the resources utilized because, while the synthesis
was based on abduction, the proof can be done in a deductive way.

In the concrete SWS Mediation Scenario, a final goal for the SWS Challenge Media-
tion is to transform a PIP3A4PurchaseOrderRequest into a PIP3A4PurchaseOrderCon-
firmation. As this is a rather complex thing to do for a single on-the-fly agent (planner),
this goal is split up into several subgoals that are solved individually. Collectively, they
yield the desired solution.
3 The GEM approach

3.1 Situation Description.

The prerequisite for establishing a GEM system is that an enterprise’s current data, called “situation”, and its IT systems’ capabilities, e.g., Web service operations, are specified using the event calculus, e.g.

\[
\text{Situation: } [\text{por(}\text{por1})] \quad \text{Operation: } [\text{createOrder(POR)}, \\
\quad \text{requires,}[\text{por(POR)}]), \\
\quad \text{terminates,}[]], \\
\quad \text{initializes,}[\text{po(PO)},\text{po_basedOn(PO,POR)}])
\]

3.2 GEM Agents.

A GEM agent is a generic software with a memory that observes the enterprise’s situation, and influences it through operation invocations. An agent is configured with respect to the types of relevant observations, the business goals to pursue upon an observation, and the business execution policies to maintain when influencing the situation.

The processing of a GEM agent is structured into the five components Analyzer, Goal Hypothesizer, Guide, Performer (comprising the Execution Initiator, and the Execution Monitor), Explainer depicted in Fig. 1. The components are realized as PROLOG programs. They store their internal results in the memory of the agent. Concretely,

- The Analyzer probes the situation based on the configured relevant observation, e.g., reacting on a purchase order request \( ?-\text{por(POR)} \), and stores the observations, e.g., \( \text{por(}\text{por1}) \).

- The Goal Hypothesizer generates a business goal based on both the configuration and the observation, e.g., the goal is to trigger the internal purchase order management denoted by a purchase order document in the situation, and the confirmation of the purchase order request denoted by a purchase order request document in the situation \( \text{po(}\text{pol1}),\text{poc(poc1)} \).

- The Guide invokes an event calculus planner [7] with the inputs of the IT system’s capabilities, the situation, the business execution policies, and the goal, resulting in a partial-order plan, e.g.,

\[
[[\text{happens(}\text{createOrder(}\text{por1}),t1), \\
\quad \text{happens(}\text{sendFeedback(}\text{por1}),t2)], \quad [\text{before(t1,t2)}]]
\]

- The Execution Initiator creates a process execution instance in the memory of the agent based on a plan, e.g., stating that the execution currently stands before the first operation’s time \( (\text{before,}t1) \). In addition, the Execution Initiator tells the Analyzer to wait for the precondition of the first operation in the plan to occur.

- The Execution Monitor only reacts on observations made by the Analyzer that were requested by the Execution Initiator. This is in contrast to the Goal Hypothesizer, which reacts only to observations made due to the agent’s scope configuration.
The Execution Monitor’s processing depends further on the observation made. If a precondition was observed, the respective operation is invoked in the IT system, e.g., by a Web service call, and the process execution instance is updated, e.g., \(((after,t1))\). In this case, the Analyzer is asked to wait for the postcondition of the just invoked operation. If a postcondition was observed, the process execution instance is updated, e.g., \(((done,t1),(before,t2))\), and the Analyzer is asked to wait for the postcondition of the respective operation.

Partial order plans are correctly tracked by triggering multiple preconditions when the postcondition of an operation preceding a fork is observed, and only invoking an operation following a join when all operations preceding the join are known to be finished.

### 3.3 Hiding the Agent Structure.

As shown in Fig. 4, several GEM agents can be hierarchically encapsulated inside the jABC within a single GraphSIB each, and thus their complexity and internal organization can be hidden from the end users. The overall process becomes a black box, and could be hosted itself as a service on some remote machine.

### 4 Using GEM to solve the Mediation

This generic GEM Agent provides a generic on-the-fly synthesis capability. Its implementation is completely independent from the concrete setting and problem to be solved. Its behaviour is in fact controlled (and this way parameterized) by the knowledge basis, in particular by the external policies that describe the enterprise system under consideration.

We use GEM’s Prolog reasoner to 1) derive specific subgoals and 2) call Moon’s web services. This is a problem-oriented decomposition, since the specification of the
Web service semantics, the enterprise’s goals and policies happens on a higher level not mentioning concrete operations and their permissible sequences, but rather talking about desired and unwanted dependencies of the business objects manipulated through the operations using the event calculus (see below), that takes here the role played by SLTL in the companion approach.

As such, the entire GEM approach is embedded within the jABC, and the synthesis process is modeled in the jABC.

4.1 Integrating the GEM system in jABC.

To use the Prolog implementation of GEM in the Java based jABC, a wrapper class encapsulates the calls of any GEM agent procedure: Situation Analyzer, Goal Hypothesizer, Guide, Execution Initiator and Execution Monitor. This is the same technique we used in [5] to call the LTL synthesis algorithm, which is implemented in C++ and runs on a remote machine.

Each agent procedure is embedded in an own SIB in the jABC, so that this way we have a GEM SIB-palette in the jABC and we can call every agent procedure individually. The orchestration of the GEM agent procedures is modelled in the jABC as a service logic graph: its workflow is presented in Fig. 1. The GEM system is executed on a remote machine and called via web services, so GEM users need not take care of any Prolog installation: this is virtualized by the jABC.

4.2 The GEM Synthesis Process in jABC.

In every agent execution, the Analyzer is the first procedure to be executed (thus the start SIB, underlined): here the situation is observed. As explained in Sect. 3, the Analyzer decides that a new plan has to be created by calling the Goal Hypothesizer or to follow the plan execution by calling the Execution Monitor. Alternatively the Analyzer’s observation can end up with the result that nothing is done for the agent. So the three SIBs Analyzer, Execution Initiator, and Execution Monitor are legal final SIBs for the GEM orchestration graph. Additionally, every SIB can terminate the workflow in case of an error. In case of the observation of an exceptional pattern, first the Execution Monitor is called to reset the execution to the last goal, and then the Goal Hypothesizer is called to create a new plan.

4.3 Service invocation: The invocation of a (Moon) Operation.

When the Execution Monitor requests an (abstract) operation, a concrete service must be called. Solving the problem in general turns out to be more complex than first expected. Besides the communication of the invocation request itself (service invocation), jABC has to manage storing the data and mapping between the data representation of GEM and the real world (service binding and communication).

To communicate an invocation request, the SIB wrapper offers an own Java class, Invoke, containing the name of the operation, a list of input identifiers, a list of output identifiers and finally an operation identifier. The operation identifier is used by
jABC to call the synchronous response method, also offered by the interface. Using this method, jABC informs GEM about the success of the operation.

As we can see, GEM and jABC only exchange identifiers for the real world objects, like input and output values. The advantage is that GEM does not need to store complex objects nor send complex messages. So the planner only has to include and manage the (abstract, minimal) information needed for decision making. This decouples GEM from concrete data issues.

On the other hand we need to map the GEM id’s to the real world data, and of course store the data and make it accessible during execution. This is usually done via the internal execution context of the jABC. However, to enable other (external) systems to enhance the situation for GEM with further knowledge and to store it, we introduce a similar external execution context, which is accessible via web services. As shown in Fig. 2, for every requested data, like SearchCustomerResponse or LineItem, an own getter and setter service exists, to load and save the data using the id’s assigned by GEM with the Invoke-object.

Two generic DataLoad SIBs were created, respectively to Load the input data into the jABC context of the GEM agent execution, and to Save it from the jABC to the external context. These SIBs must know which context-ids to use when accessing the jABC context. To this aim, the signatures of all operations on which the GEM operations and the Operation-SIBs in jABC are based upon are initially imported, and their parameter names are then taken to load/save the objects from/to the jABC context.

One more SIB is necessary to guarantee a completely generic GEM agent workflow: the Operation Call Switch. This SIB simply forwards the invocation to the right Service operation, on the basis of a description of the Operations supported by an external agent. Fig. 3 shows the exemplary invocation request of the Moon Service Search Customer, together with the operation selection Operation Call Switch and the Load/Save SIBs.

Once an operation is invoked, the Operation Call Switch is called again: since GEM may work with multiple threads, the Execution Monitor could send more than one invocation request in one message. The Operation Call Switch then decides to call the next operation or, if this was the last operation, return control to the next agent.
4.4 GEM at work for the Mediator.

To remain close to the Mediation scenario description, two Moon agents are created:

- a CRM agent that communicates with Moon’s customer relationship management system, and
- a Sales agent that communicates with Moon’s Order Management System.

Since jABC supports hierarchical Service Logic Graphs, we hierarchically encapsulate the entire GEM agent graph in a single GraphSIB: this way both agents use the same model of Fig. 1, which is instantiated twice. A model parameter determines the type of the agent.

As we see in Fig. 4, when a Blue Purchase order request arrives, the SIB SaveAuthToken starts the planning of the mediator solution. We have two instances of the GEM agent of Fig. 1, one for the CRM Agent and one for the Sales Agent, each of them determining

- either the next operation to be executed by that agent, passing the control to the own operation executor (SIBs CRM Agent Operations and Sales Agent Operations)
or the own termination, passing then the token to the next agent.

During execution, one agent is thus able to complete one step of its execution, then passing the execution to the next agent. This rotation prevents agents from taking a lot of time completing multiple tasks, while other agents wait. In fact, at any time the situation could be enriched by another participant, giving an agent new tasks and inducing longer delays for the other agents, up to starvation. The scheduling strategy can however be easily changed by modifying this graph.

The set of operations of an agent is itself modelled within a GraphSIB, see Fig. 4. While Fig. 3 shows the only operation (SearchCustomer) of the CRM agent, the Sales agent has much more to do. As shown in Fig. 5 the sales agent has to call the four web service operations CreateOrder, AddLineItem, CloseOrder, and SendPOConfirmation. Additionally the SIB SendWaitFor is called to be able to react on an asynchronous response, here used by the service which receives the confirmation of any line item.

Each of these operations is implemented along the lines shown in Fig. 3: it is in fact an invocation pattern comprising a Load-SIB, potential SIBs for data mediation, the specific Operation-SIB, a Save-SIB and a SIB for the synchronous response.

4.5 Mediator Execution: Communication with the testbed.

To communicate with the SWS Challenge testbed, two more web services are created:

- The first receives the PurchaseOrderRequest message of the Blue Company, writes the relevant data into the GEM situation and stores the corresponding objects into the external context. This is the only place where the situation is written outside the GEM system.
- The second web service receives the confirmation of any line item and informs GEM about it. The receipt of a line item confirmation is designed using a procedure for an asynchronous response of the AddLineItem operation. The interface
offers a further method for that purpose and an additionally class WaitFor (similar to invoke), with which GEM informs jABC that the agent waits for a specific operation to be done. So every operation can have a synchronous and asynchronous response.

5 Conclusion

We showed here how to reuse the framework, the setting, and the semantic modelling for the automated synthesis of the SWS Challenge Mediator presented in [5], in order to exchange the synthesis paradigm from a linear-time-logic proof based algorithm to an abductive synthesis implemented in Prolog, that responds directly to situational changes.

Both synthesis approaches have the potential to become a highly flexible middle layer for large scaling business processes, combining the advantages of agent technology and service-oriented computing. By enhancing the underlying reasoning technology the prototype has the chance to become a declarative mediator controllable by non-programmers, like business experts. Since it is situation aware, it can react promptly to situational changes, and this way potentially self-adapt during execution.

The implementation is still prototypical, and restricted so far to Mediation scenario Part 1. We are currently working on extending it to the other parts of the scenario, and on a hybrid approach that takes advantage of the dynamic agent behavior within a full service-oriented framework that considers even the reasoning algorithms as retargetable services.

References

Data and Process Mediation Support for B2B Integration

Maciej Zaremba\textsuperscript{1}, Maximilian Herold\textsuperscript{1}, Raluca Zaharia\textsuperscript{1}, and Tomas Vitvar\textsuperscript{2}

\textsuperscript{1} Digital Enterprise Research Institute
National University of Ireland in Galway, Ireland
firstname.lastname@deri.org

\textsuperscript{2} Semantic Technology Institute (STI2)
University of Innsbruck, Austria
tomas.vitvar@sti2.at

Abstract In this paper we present how Semantic Web Service technology can be used to overcome process and data heterogeneity in a B2B integration scenario. While one partner uses standards like RosettaNet for product purchase and UNIFI ISO 20022 for electronic payments in its message exchange process and message definition, the other one operates on non-standard proprietary solution based on a combination of WSDL and XML Schema. For this scenario we show the benefits of semantic descriptions which are used within the integration process to enable rule-based data and process mediation of services. We illustrate this dynamic integration process on the WSMX – a middleware platform conforming to the principles of a Semantic Service Oriented Architecture.

1 Introduction

Inter-enterprise integration is the essential requirement for today’s successful business. While technologies around RosettaNet, EDI or ebXML certainly brought new value to inter-enterprise integration, its rigid and hard-wired configuration makes it still difficult to reconfigure, reuse and maintain. In addition, cooperating partners often use different Business-to-Business (B2B) standards thus either adoption of a standard used by a “stronger” partner or maintaining more than one B2B standards within one B2B integration is required.

Semantic technologies offer promising potential to enable B2B integration that is more flexible and adaptive to changes that occur over a software system’s lifetime [8]. Semantic Web services (SWS), by augmenting services with semantic descriptions, is one of the candidate technology for more automated and dynamic service integration and discovery. Semantically annotated services promote the integration process by enabling runtime data and process mediation. The scenario used in this paper is based on the requirements for the SWS Challenge\textsuperscript{3}, and in particular on a scenario for data and process mediation. In

\textsuperscript{3} http://sws-challenge.org
comparison to the previously developed mediation solution [3,4] we have pro-
vided a support for a new electronic payment scenario and our solution has been
extended with a fully-fledged rule-based data mediation [6].

In order to address the SWS challenge requirements, we base our solution on
the specifications of WSMO[9], WSML[9] and WSMX[12] providing a concep-
tual framework, ontology language and architecture for Semantic Web Services.
The overall contribution of our work is to show: (1) how flat XML schema of
RosettaNet, UNIFI ISO 20022, and other messaging schema used by different
partners can be semantically enriched using the WSML ontology language, (2)
how services provided by partners could be semantically described as WSMO
services and built on top of existing systems, (3) how conversation between part-
ners and their services can be facilitated by the WSMX integration middleware
enabling semantic integration, and (4) how generic, rule-based data and process
mediation can be applied between heterogeneous services within the integration
process.

2 Solution Architecture

In SWS-Challenge mediation scenario there are two business partners (Moon and
Blue) involved that need to have their systems integrated using semantically-
enabled technology. The scenario describes how Moon has signed agreements to
exchange purchase order messages with its client company called Blue using the
RosettaNet PIP 3A4 specification. Details of our solution to RosettaNet scenario
has been previously described in [3].

In this paper we provide a support for the new payment scenario and we
extend our previous solution with a support for fully-fledged rule-based data
mediation. We have resolved technical problems with integrating rule-based Data
Mediation component and we showcase a generic data mediation solution. We
build our solutions on the SWS framework based on the WSMO (Web Ser-
vise Modeling Ontology [9]) conceptual model, WSML (Web Service Model-
ing Language [9]) language for service modeling, WSMX (Web Service Execu-
tion Environment [12]) middleware system, and WSMT (Web Service Modelling
Toolkit [5]) modelling framework. In order to model the scenario, we use WSMO
for modeling of services and goals (i.e. required and offered capabilities) as well
as ontologies (i.e. information models on which services and goals are defined)
all expressed in the WSML-Flight ontology language. WSML-Flight provides a
Datalog expressivity extended with inequality and stratified negation that is suf-
ficient for addressing requirements of SWS Challenge scenarios. We use KAON2
reasoner and IRIS for the inference over WSML-Flight ontologies.

---

5 http://kaon2.semanticweb.org
6 http://iris-reasoner.org
2.1 Payment Scenario

In Figure 1, the global architecture of our solution for the case scenario is depicted. The whole integration process of the Blue and Moon companies happens in two phases: (1) integration setup phase and (2) integration runtime phase. During the setup phase, the integration ontologies are designed including the models used in UNIFI ISO 20022 payment information and the models used by Moon’s financial systems. The design and implementation of adapters, creation of WSMO ontologies and services, rules for lifting/lowering, mapping statements between used ontologies and registration of ontologies, services and mapping statements with WSMX are also carried out. During the runtime phase, interactions between Blue and Moon systems are executed.

In order to address integration of Blue and Moon companies, our goal is to use Semantic Web service technology to facilitate conversation between all systems, to mediate between the data used by Moon, as well as to ensure that the message exchange between all parties is correctly choreographed. Data Mediation was not necessary in payment scenario as most of the integration process was carried out on the Blue side where homogenous ontology of UNIFI ISO 20022 electronic payment information has been used. Due to the simplicity of Moon’s data in this scenario there was no need to provide data mediation support and bank account details provided by Blue service has been directly utilized in integration process. Process mediation is involved in mapping of message exchanges defined by UNIFI ISO 20022 process to those defined in the WSDL of the Moon back-end systems. Conversation between systems including data and process mediation operates on semantic descriptions of messages, thus transformation from messages used by existing systems to ontological level is first performed.

![Figure 1. Global Integration Architecture](image)

In our solution, we built the integration between the Blue and Moon systems on the WSMX platform which resides between Moon and Blue infrastructure al-
ollowing the seamless integration of all involved systems. XML messages between
the partners are lifted to semantically-enabled WSML level. Blue’s message initi-
ating B2B interaction is translated into WSMO Goal what allows for goal-driven
discovery, service execution and mediation that is provided by WSMX environ-
ment. Goals describe requirements over the service to be discovered and are
specified independently from the actual service. The following basic blocks are
involved in our solution to SWS-Challenge B2B integration:

- **Existing Systems.** The existing systems are Moon’s back-end service Fi-
nancial Information Provider as well as Blue’s UNIFI ISO 20022 payment
system with Accounting Department System and Management Department
System. Each system communicates using different formats, e.g. Blue’s sys-
tems communicates according to the UNIFI ISO 20022 messages (Payment
Information), whereas communication with the Moon’s system is more pro-
prietary - specified in their WSDL descriptions. Detail descriptions of these
WSDL interfaces can be found at SWS challenge web site.

- **Adapters.** In order to connect existing systems with WSMX, adapters are
used to mediate between the different communication protocols and lan-
guages. Since WSMX internally operates on the semantic level handling
messages in WSML, adapters facilitate lifting and lowering operations al-
lowing message to be transformed from XML to WSML and vice-versa. The
adapter also handles the application logic of identifying a valid Goal to be
sent to the WSMX for the incoming message and subsequently sending the
lifted form (WSML) of the purchase order message. Goal-based invocation
is the basis for advanced semantic discovery and mediation. In Figure 1, the
UNIFI ISO 20022-WSMX and Moon-WSMX adapters are used for connec-
tion to the Blue and the Moon system.

- **WSMX.** WSMX is the integration platform which facilitates the integration
process between different systems. The integration process is defined by the
execution semantics describing interactions of middleware services including
discovery, mediation, invocation, choreography, repository services, etc. De-
tail descriptions of execution semantics and middleware services for our use
case is given later in this section.

A payment request is sent from the client in XML to the entry point of
UNIFI ISO 20022-WSMX adapter. In the UNIFI ISO 20022-WSMX adapter, the
message captured in XML is lifted to WSML according to the UNIFI ISO 20022
ontology and rules for lifting. The abstract WSMO goal\(^7\) is created including
definitions of requested capabilities and a choreography. Requested capabilities
describe the desired capability of the requester (Blue company) used during the
discovery process whereas goal choreography describes how the requester wishes
to interact with the environment. Since a WSMO service is, from the WSMX
point of view, represented by an adapter (the adapter can be understood as a
wrapper around existing application – in our case Blue’s RosettaNet system),
the choreography here reflects the communication pattern of the adapter (hence

\(^7\) We refer to the abstract goal as a goal which contains no instance data (input values)
it does not include interactions regarding acknowledgments of messages). After
the goal is created, it is sent as a WSML message to the WSMX environment
through the AchieveGoal entrypoint.

The WSML message is passed through the Communication Manager to the
execution semantics which again first parses the data into the memory object by
invoking the WSMX Parser. In general, more independent conversations can be
running inside WSMX, thus information carried by the context is used to identify
the execution semantics associated with the conversation from the context. The
execution semantics then passes obtained data to the WSMX Process Mediator.

The role of the WSMX Process Mediator is to decide, which data will be
added to which choreography, i.e. requester’s or provider’s choreography. Please
note that choreographies of WSMO services are modeled as Abstract State Ma-
chines [1] and are processed using standard algorithms during runtime. Memory
of the choreography contains available instance data of ontological concepts. A
choreography rule which antecedent matches available data in the memory is se-
lected from the rule base and by execution of the rule’s consequent, the memory
is modified (data in the memory is updated, deleted or removed). This decision
is based on analysis of both choreographies and concepts used by these chore-
ographies and is in detail described in [2]. In our scenario, Process Mediator first
updates the memory of the requester’s choreography with the information that
the Payment Request has been sent. The Process Mediator then evaluates that
data should be added to the memory of the provider’s choreography.

**Choreography Process.** Figure 2 depicts Blue’s Payment choreography in-
cluding rules that are elaborated further on listings provided in this section.
First, a controlled instance is initialized during the execution and can be modi-
fied only by the choreography execution. Its value attribute belongs to a finite set
of states that are used to control the execution. Each rule checks in its condition
the controlled instance and is fired only when controlled instance permits.

The rules in Moon Payment choreography specify: a set of variables, a rule
conditions for the variable binding and a set of actions operating on the data
provided in the variable bindings when rule conditions are satisfied.

The following notation is used in Moon Payment choreography pseudocode:
keywords in WSML are marked with bold; “?” followed by an identifier represents
a variable; ontology concept names are written in camel case.

**Moon’s Financial Information.** During the execution, when the first con-
dition is met (i.e. PaymentInitiation has been sent with the goal), the actions
of the rule 1 can be executed. The new BankingInformationRequest instance is
an input to the Moon’s Financial Information Provider service, the invocation
of which results in banking information (Moon’s bank account details) response
message.

Listing 1.1 shows rule 1 in full WSML syntax. However, for the brevity we
present the rest of the rules in more concise, pseudocode form as shown on
Listing 1.2.
Figure 2. Blue’s Payment Scenario Choreography

Listing 1.1. Rule Creating BankingInformationReq in WSML Syntax (Rule 1)

forall (?controlled, ?request) with (?
?controlled{oasm#value hasValue oasm#InitialState} memberOf oasm#ControlState and
?request memberOf pay#PaymentInitiation)
do
add([#moon#hasRequestId hasValue "token_id"] memberOf moon#BankingInformationRequest)
delete(?controlled{oasm#value hasValue oasm#InitialState})
add(?controlled{oasm#value hasValue oasm#State1})
endForall

Listing 1.2. Rule Creating BankingInformationReq in Pseudocode (Rule 1)

forall PaymentInitiation ?request do
create BankingInformationRequest instance
Blue’s Accounting Department System Payment Initiation. The new PaymentInitiationFDRequest instance is an input for the Blue’s Financial Information Department System service and after the invocation the payment initiation response is received. The rule that triggers Blue’s Financial Department System invocation is presented on Listing 1.3.

```
forall PaymentInitiation ?request, BankingInformationResponse ?variable do
create PaymentInitiationFDRequest instance
```

Listing 1.3. Rule Creating PaymentInitiationFDRequest (Rule 2)

If the initial amount is small, the Accounting Department will accept the payment directly and there will be sufficient information to create the final response for the customer and final rule 5 can be fired finishing the choreography execution.

Blue’s Management Department System First Authorization Request. If Blue’s Accounting Department requires an authorization code then the authority’s name will have to be determined for the payment approval via Blue’s Management Department System service. It might be required to ask more than once for the authorisation, thus there might be many authorisation responses (some rejecting and some accepting the payment). In order to avoid checking the same data again, the service response instance is flagged as not processed (by setting the attribute to true). Also, if an authorisation request is created, the value attribute of the controlled instance is set to identify the authority that was asked to accept the payment. For example, if the payment amount is more than 10.000 Euro and less than 50.000 Euro, the third authority (Arnold Black) will be asked first and the state will be set to ”Authorised3”.

```
forall PaymentInitiationFDResponse ?variable with response ?code attribute and ?controlled instance
create PaymentStatus instance with code "PI_ACCEPTED"
if (?code = "AUTHREQUIRED") then
  if(?amount > 1999.99 and ?amount < 3000) then
    create PaymentAuthorizationMDRequest instance with the first authority’s name
    add(?controlled[value = "Authorised1"])
  end
  if(?amount > 49999.99) then
    create PaymentAuthorizationMDRequest instance with the last authority’s name
    add(?controlled[value = "Authorised4"])
    add notProcessed attribute to the response of the service with the value "true"
```

Listing 1.4. Rule Determining Authority Required for MDS Payment Authorization (Rule 3)

Blue’s Management Department System Subsequent Authorization Requests. The rule selects the response instances from the Management Department System service that were not checked yet. If the Management Department System accepts the payment, it will provide an authorisation code, which is used to create a request to the Blue Accounting Department System. Otherwise the request for payment authorisation has to be repeated with higher-rank authority. We can determine which authority was previously asked by looking at the value attribute of the controlled instance. For example, if the
state was "Authorised3", it means that the request to the Management Department System service will be formulated with the forth authority’s name and the state will be changed to "Authorised4". Processed response is flagged as "false" (by changing the notProcessed attribute) and (as in rule 3) the latest Management Department System response is marked as not processed. If the previously asked authority was the forth, there is no authority left to ask for payment approval, therefore the final PaymentStatus response is created with the status "PI_REFUSED_AUTH_FAILED" and the execution ends. The same situation applies if the Management Department System service fails to authorise the request.

forall PaymentAuthorizationMDResponse ?response with attributes ?code and notProcessed="true" do
  if (?code = "ACCEPTED") then
    create PaymentInitiationFDRequest instance with the authorisation code of the ?response
  if (?code = "DENIED") then
    add(?response[notProcessed = "false"])
  if (?controlled[value = "Authorised1"]) then
    create PaymentAuthorizationMDRequest instance with the second authority’s name
    add notProcessed attribute to the response of the service with the value "true"
    add(?controlled[value = "Authorised2"])
  if (?controlled[value = "Authorised4"]) then
    create PaymentStatus instance with code "PI_REFUSED_AUTH_FAILED"
  if (?code = "FAILED") then
    create PaymentStatus instance with code "PI_REFUSED_AUTH_FAILED"

Listing 1.5. Loop over Blue’s MDS Authorization (Rule 4)

Final Payment Status Response. This rule will be executed if the authorisation response in rule 4 was "ACCEPTED". There is another PaymentInitiationFDResponse instance in the state ontology as a result of executing rule 1, but its code is "AUTHREQUIRED" (otherwise the execution would have already ended) and it does not have sufficient information to create final response for the Blue client. PaymentInitiationFDResponse instance is selected since it contains header and originalGroupInfoAndStatus attributes in order to create and send the payment status response to the client.

forall PaymentInitiationFDResponse ?variable do
  create PaymentStatus instance with code "PI_ACCEPTED"

Listing 1.6. Final Payment Status Response (Rule 5)

2.2 Purchase Order Mediation Scenario - Data Mediation

Although we chose not to use the functionality of the WSMX Data Mediator in the payment SWS Challenge scenario, we see the recent integration into WSMX as a noteworthy improvement of our system compared to the previous SWS Challenge. SWS Challenge Purchase Order Mediation scenario\(^8\) is more suited to show the added value resulting from descriptive mappings between ontologies

\(^8\) http://sws-challenge.org/wiki/index.php/Scenario:_Purchase_Order_Mediation (previous solution described in [3])
(compared to hard-coded message transformations in adapters). We have successfully used the WSMX Data Mediator for this scenario and thus present the data mediation based on that example in the following paragraphs.

WSMO conceptually takes into account that there may be different implementations for data mediation. WSMO specifies OO-Mediators, which are used as descriptions that define a mediation service between two ontologies (independent of the implementation). An OO-Mediator has an ID, references to source and target ontology, and a mediationService property that points to the service or component that actually implements the data mediation between the given ontologies\(^9\). Before the choreography execution, the ontologies used in the respective choreographies of goal and web service are inspected. For each pair of goal and web service ontologies \((GO_i, WO_j)\) with \(GO_i \neq WO_j\), it is checked whether there is an OO-Mediator registered. If this is the case, the specified data mediation service is requested using the input data. Any mediation results are then combined and forwarded to choreography execution (if no data mediation was necessary or possible, the unmediated data is used).

---

**Figure 3.** Data Mediation Moon Scenario

To illustrate the example scenario, Figure 3 shows the relevant parts of the RosettaNet expected input instances as well as all of the instances that need to be created during runtime in order to communicate with the Moon legacy

\(^9\) note that OO-Mediators can also be used in other ways, which will be omitted here since it is not relevant to this example
systems. The hierarchy shown depicts instances with their respective attributes, whereas an attribute is member of a concept with the same name except denoted otherwise. The creation of the instances and attributes marked with an asterisk (*) is not the responsibility of data mediation but of choreography execution, since some of the attributes do not have any correspondence in the input data.

The WSMX Data Mediator itself uses mapping descriptions to implement the data mediation. The mappings between the RosettaNet PIP 3A4 Purchase Order Request and Moon ontologies are created during the integration setup phase. They are represented in an abstract, ontology mapping language. The creation of those mappings is a semi-automatic process (due to the requirement of accuracy). The domain expert is aided in this step by a graphical mapping tool. Utilizing different perspectives on source and target ontologies allows for the creation of complex mappings using only a simple operation, map. A contextualization strategy as well as lexical and structural suggestion algorithms provide further support for the domain expert. The model is formally described and linked to the Abstract Mapping Language (described in [10] and elaborated in [11]). Statements in the Abstract Mapping Language include, amongst others, classMappings, attributeMappings, classAttributeMappings and various conditional statements.

The mappings for this use case can be created using only the PartOf perspective of the mapping tool, which focuses on concepts, attributes and attributes’ type hierarchies. The generated mapping statements between the RosettaNet BusinessDescription and the Moon SearchCustomerRequest are shown in Listing 1.7 using the Abstract Mapping Language.

During the integration runtime phase, the Abstract Mapping Language statements are converted to WSML rules which specify the conditional creation of instances of the target ontology. The input instances and the rules are registered with a reasoner, along with the source and target ontologies. By querying the knowledge base for instances of the target ontology, the rules fire and thus generate the respective instances of the target ontology. For the given example, the knowledge base is shown in Listing 1.8 (rules and input instances only), the queries and the resulting mediated instances in Listings 1.9 and 1.10. All of these listings are in WSML.

Listing 1.7. Data Mediation Mapping Rules (simplified)

```
classMapping(BusinessDescription, SearchCustomerRequest)
attributeMapping([(BusinessDescription) businessName => string], [(SearchCustomerRequest) searchString => string])
classMapping(string, string)
```

### Listing 1.8. Knowledge Base (simplified)

```
axiom m#ccMappingRule4 definedBy
   m#mappedConcepts(rosettacore#BusinessDescription,moon#SearchCustomerRequest,?X3)
   and m#mediated1(??X3,moon#SearchCustomerRequest) memberOf moon#SearchCustomerRequest
   -: ?X3 memberOf rosettacore#BusinessDescription.
```

axiom m#aaMappingRule8 definedBy
   m#mediated1( ?X5, moon#SearchCustomerRequest ) [ moon#searchString hasValue ?Y6 ] memberOf
   moon#SearchCustomerRequest
   ...
   ?X5[ rosettacore#businessName hasValue ?Y6 ] memberOf rosettacore#BusinessDescription
   and
   ?X5 memberOf 75C7
   and
   m#mappedConcepts( 75C7, moon#SearchCustomerRequest, ?X5 ).

axiom m#ccMappingRule2 definedBy
   m#mappedConcepts( _string, _string, ?X1 )
   ...
   ?X1 memberOf _string.

instance in#input_BusinessDescriptionFromRolePartDe memberOf rosettacore#BusinessDescription
   rosettacore#businessName hasValue "Blue Company"

Listing 1.8. Data Mediation Knowledge Base

// Query 1:
?x memberOf moon#SearchCustomerRequest
// 1 result:
?x=m#mediated1(in#input_BusinessDescriptionFromRolePartDe,moon#SearchCustomerRequest)

// Query 2 (created based on the result of Query 1):
m#mediated1(in#input_BusinessDescriptionFromRolePartDe, moon#SearchCustomerRequest) [ ?y hasValue ?z ] memberOf moon#SearchCustomerRequest and
?z memberOf ?avC.
// 1 result:
?y=m#searchString, ?z="Blue Company", ?avC= "http://www.wsmo.org/wsml/wsml−syntax#

Listing 1.9. Data Mediation Queries

instance m#mediated1(’in/input_BusinessDescriptionFromRolePartDe, moon/SearchCustomerRequest’) memberOf rosettacore#BusinessDescription
   rosettacore#businessName hasValue “Blue Company”

Listing 1.10. Data Mediation Final Result

During choreography execution, the additionally required instances and attributes not generated by data mediation are added. Listing 1.11 shows the respective transition rule adding an authToken attribute to the Moon SearchCustomerRequest and CreateOrderRequest.

forall { ?controlstate, ?searchCustReq, ?createOrdReq } with {
   ?controlstate[oasm#value hasValue oasm#InitialState] memberOf oasm#ControlState
   and
   ?searchCustReq memberOf moon#SearchCustomerRequest
   and
   ?createOrdReq memberOf moon#CreateOrderRequest
} do
   add( [ ?searchCustReq[ moon#authToken hasValue “MaciejZaremba” ] ] ) // add authToken
   add( [ ?createOrdReq[ moon#authToken hasValue “MaciejZaremba” ] ] ) // add authToken
   add( [ ?controlstate[ oasm#value hasValue oasm#CreateOrder ] ] ) // add authToken
   delete( [ ?controlstate[ oasm#value hasValue oasm#InitialState ] ] )
endForAll

Listing 1.11. Choreography Example

More information on the WSMX Data Mediator can be found in [6] and [7].

3 Conclusion

In this paper we presented our approach to dynamic B2B integration based on the Semantic Web Services technology in particular we have addressed an
extended mediation scenario and provided a generic, rule-based data and process mediation of heterogeneous services. Our solution is a contribution to the SWS Challenge and further, will be part of an evaluation within this initiative.

Acknowledgments

This work is supported by the Science Foundation Ireland Grant No. SFI/02/CE1/I131, and the EU projects SUPER (FP6-026850), and SemanticGov (FP-027517).

References

SWSCChallenge workshop at ISWC 2008

Karlsruhe, Germany
Model-driven Service Integration using the COSMO framework

Dick A.C. Quartel¹, Stanislav Pokraev¹, Teduh Dirgahayu², Rodrigo Mantovaneli Pessoa² and Marten van Sinderen²

¹ Telematica Instituut, P. O. Box 589, 7500 AN, Enschede, The Netherlands, {Dick.Quartel, Stanislav.Pokraev}@telin.nl
² Center for Telematics and Information Technology (CTIT), University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands, {T.Dirgahayu, MantovaneliR, M.J.vanSinderen}@ewi.utwente.nl

Abstract. In this paper, we describe an approach for solving the integration problem in the Purchase Order Mediation scenario of the Semantic Web Service Challenge². The key feature of our approach is that service models are employed at different abstraction levels to develop end-to-end integration solutions from business requirements to software implementation.

Keywords: Service mediation, MDA, model transformations.

1 Introduction

The goal of the SWS Challenge is to explore the trade-offs of various existing technologies that aim at automation of mediation, choreography and discovery of Web Services. For that reason, the SWS Challenge defines a number of scenarios providing a standard set of problems, based on industrial specifications and requirements.

In this paper, we present a model-driven approach for solving the integration problem described in the Purchase Order Mediation scenario of the SWS Challenge. Model-driven techniques are used to abstract the integration problem and solution to a higher (platform-independent) level. This way, the problem and solution can be captured in a technology independent manner enabling more active participation of business domain experts.

This paper is structured as follows: Section 2 briefly presents our integration framework. Section 3 shows how we use the framework to solve the integration problem from the Purchase Order Mediation scenario. Section 4 compares our work with the solutions provided by the other SWS Challenge participants. Finally, section 5 presents our conclusions and future work.

¹ The presented work has been done in the Freeband Communication project A-Muse (http://a-muse.freeband.nl). Freeband Communication (www.freeband.nl) is sponsored by the Dutch government under contract BSIK 03025
² http://sws-challenge.org
2 Integration Framework

We approach the design of a mediator as a composition problem: each service that is requested by some system has to be composed from one or more services that are provided by one or more other systems. Fig. 1 illustrates this for the case of two systems A and B. Mediator M offers a mediation service that matches requested service S1 of A by composing services S3 and S4 that are offered by B. The mediator should provide such a mediation service for each service that is requested by A (and B).

![Fig. 1. Service mediation as service composition.](image)

To support the design, implementation and verification of mediators we have developed an integration method. Our method uses the COSMO framework [11] to model and reason about services. The method defines a number of steps to build end-to-end integration solutions and to verify their correctness. In the following sections, we briefly describe the COSMO framework and the steps of our method.

2.1 The COSMO framework

We define a service as “the establishment of some effect (or value) through the interaction between two or more systems”. Based on this definition, the COntceptual Service MOdeling (COSMO) framework defines concepts to support the modeling, reasoning and analysis of services.

We distinguish four service aspects, i.e., information, behavior, structure and quality, representing categories of service properties that need to be modeled. The structure aspect is concerned with modeling the systems that provide or use services, and their interconnection structure. The interconnection structure comprises (amongst others) the interfaces at which services are offered. The behavioral aspect is concerned with the activities that are performed by systems as well as the relations among these activities. The information aspect is concerned with modeling the information that is managed by and exchanged among systems. The quality aspect is concerned with modeling the non-functional characteristics of services. These qualities often play an important role in the selection of services. Examples of quality aspects are the “cost” associated with a service or the “response time” of a service.

Besides service aspects, we distinguish three generic abstraction levels at which a service can be modeled, namely, goal, choreography and orchestration level. A model at goal level describes a service as a single interaction, where the interaction result represents the effect of the service as a whole. A model at choreography level refines the model at goal level by describing the service as a set of multiple related, more concrete interactions. A model at orchestration level describes the
implementation of the service using a central coordinator that invokes and adds value to one or more other services.

Finally, we distinguish different roles of the systems involved in a service: the user, provider and integrated role. The integrated role abstracts from the distinction between a user and provider by considering interactions as joint actions, thereby focusing on what the user and provider have in common.

This paper mainly considers choreographies and orchestrations from the behavior and information aspect, and by distinguishing between a user and provider role. Furthermore, services are modeled close to the level at which they are described using WSDL, while abstracting from technology details. Therefore, and for brevity, we only explain COSMO’s operation concept below and its notation using ISDL [10]. For an overview and explanation of the COSMO concepts, we refer to [11].

Fig. 2 (i) and (ii) depict the operation concept and its interpretation in terms of a flow chart-like notation, respectively. An operation represents a composition of three instances of message passing: the sending (invoke) and receipt (accept) of an invocation, followed by either the sending (reply) and receipt (return) of the invocation result, or the sending (fault) and receipt (catch) of a fault message. The use of the reply-return and the fail-catch message passing instances are optional, i.e., either one or both parts may be omitted; e.g., to model one-way operations.

2.2 Integration method

The steps of our integration method are depicted in Fig. 3. For the sake of readability, we consider two systems, but the same steps apply to the case of multiple systems.

In Step 1 of our method, we derive the platform-independent models (PIMs) of the services to be integrated by abstracting from all technical details in the platform specific models (PSMs). Next, in Step 2 we increase the coverage and precision of the PIMs by adding semantic information that cannot be derived from the original service
descriptions (PSMs). In Step 3, we solve the integration problem at PIM level, which enables the more active participation of domain experts. In addition, the abstract nature of the integration solution allows one to reuse it for different implementation technologies. Besides, the semantically enriched service models allow some integration tasks to be fully or partially automated. Next, in Step 4 we verify the correctness of the integration solution using various analysis techniques. Finally, in Step 5 the service PIM is transformed to a PSM solution by mapping the integration solution to a specific service computing platform.

3 Application of the integration framework

This section presents the application of our framework to the Purchase Order Mediation scenario of the SWS Challenge. For this purpose, the integration method is made concrete by deciding on, amongst others, the type of PSMs that are considered, the languages to be used at PIM level, and related to these choices the transformations and analysis techniques that are needed, e.g. have to be developed.

Step 1: Abstract service PSMs to Service PIMs. In the first step, we derive the platform independent information and behavior models of the services of Blue and Moon, which are specified in WSDL. ISDL [10] is used to represent the service behavior, and UML class diagrams are used to represent the information models.

This step is automated using the WSDL import function of the Grizzle tool [4]. Grizzle is an integrated editor and simulator for ISDL, and uses Java to represent and execute operation parameter constraints. Once a WSDL document is imported, a behavior model is generated that represents the user or provider role of the web service, in terms of operation calls or operation executions, respectively. In addition, an information model is generated consisting of Java classes that represent the information types that are referred to by the operations in the behavior model. The transformation from WSDL to ISDL and Java is implemented using JAXB and JAX-WS [5]. We use an EclipseUML tool [3] to represent (and manipulate) the information model using UML class diagrams.

Step 2: Semantic enrichment of PIMs. The WSDL descriptions of the example scenario define the services that are provided by Blue, Moon and the Mediator, in terms of their operations and the types of the input and output messages of these operations. However, WSDL does not define the interaction protocols of the involved systems, i.e., the possible orderings of the operations. Therefore, to derive the complete PIMs of Moon and Blue, we have to use and interpret the provided textual descriptions. This is a manual process.

Firstly, the behavior models that were generated in Step 1 are completed by defining relations between operations. These relations can be derived from the scenario description. This includes the explicit modeling of the repetitive process of adding and confirming line items. Fig. 4 depicts the enriched model of the service requested by Blue and the service provided by Moon OM.

Secondly, the information model may be enriched by interpreting the scenario description. A WSDL description defines the syntax of the messages that are exchanged, but does not provide information about their semantics. This semantics can be made explicit by defining new classes, properties and relations among classes.
Furthermore, the meaning of classes and their properties may be defined by a mapping onto some domain-specific ontology, e.g., the Universal Data Element Framework [15]. The benefits of these types of semantic enrichment can however, only be fully exploited when using an ontology language (such as OWL [16]) that allows one to explicitly model and reason about the semantics of classes and their properties. The semantic analysis of information models is considered in the next version of our integration method.

![Fig. 4. Enriched behavior models of Blue and Moon OM](image)

**Step 3: Design of the mediator PIM.** In this step, we design the behavior and information model of the Mediator. The information model of the Mediator is constructed from the union of the information models of Blue and Moon. For the same reason as explained at the end of the previous section, this information model is not enriched to define the relationships between the classes and properties from the information models of Blue and Moon, except for informal annotations that may explain these relationships using natural language. The information model is extended, however, with classes to represent status information of the Mediator, such as the set of order line items that have been confirmed so far.

The construction of the behavior model of the Mediator requires the definition of (i) the services provided and requested by the Mediator, (ii) the composition of these services by relating the operations of the services, and (iii) the data transformations among the parameters of the operations.

In the example scenario, the Mediator provides one service that must match the service requested by Blue. The service provided by the Mediator can initially be defined as the ‘complement’ of the service requested by Blue. The complement of a service is obtained by changing each operation call into an operation execution, and vice versa, while keeping the same parameters. In addition, the relations among the operations and the parameter constraints may (initially) be retained. Likewise, the services that are requested by the Mediator can be obtained by taking the complement of the services that are provided by Moon. These retained relations and parameter constraints may be refined in the next design steps, respectively. For example,
relation between operations `receiveRequest` and `receiveConfirmation` has to be implemented by the orchestration of the services of Moon. As another example, the disabling relation (represented by the black diamond on top of a horizontal bar in Fig. 4) between `addLineItem` and `closeOrder` will be replaced by an enabling relation, since the order should be closed only after all line items have been added.

The design of the Mediator behavior can now be approached as the search for a composition of the requested services that conforms to the provided service. The structure of this composition is defined by the (causal) relations among the operations. Most of these relations can be found by matching the input that is required by each operation to the output that is produced by other operations. For example, operation `search` of Moon’s CRM service requires as input a search string that can be matched to some element of the customer information that is part of the purchase order information received by operation `receiveRequest`. This implies that a relation should be defined between `receiveRequest` and `search`. Fig. 5 depicts the design.

Fig. 5. Design of the mediator

Matching inputs and outputs is however insufficient to find all relations. For example, although operation `receiveRequest` and operation `search` provide information that matches the input required by operation `createNewOrder`, the information that is provided by `receiveRequest` should be used instead. This hidden assumption has to be
made explicit in the behavior model. Furthermore, specific processing logic may have to be designed manually. For example, the process of receiving confirmations from Moon’s OM system depends on information from operations receiveRequest (the items to be confirmed), createNewOrder (the order id) and addLineItem (the item id used by Moon), and depends on internal status information of the Mediator, i.e., the knowledge that operation closeOrder has occurred and the set of confirmations that has been received so far. Even when these information requirements are given, the relations involved in the repetitive processing of confirmations can not be derived easily, and have to be designed explicitly.

The definition of the data transformations among operation parameters can be approached as a refinement of the relations among operations defined in the preceding step. These relations define for each operation on which other operations it depends, and therefore which output parameters can be used in the generation of its input parameters. The data transformations then define how the value of each input parameter is generated from the values of the output parameters and, in some cases, some internal state information of the Mediator. This involves the definition of translations between the information models of Blue and Moon. However, these translations only need to address those parts of the information models of Blue and Moon that are related via the relations defined in Step 2.

To express data transformation functions we have defined a Domain-Specific Language (DSL) using the Eclipse TCS [14]. In our DSL, a data transformation is specified as a set of relations among two or more objects. In a from clause a number of variables are bound by evaluating queries on a source object. In a create clause a new (target) object is created and its properties are set to the values of the variables bound in the from clauses. Likewise, an update clause takes an existing object and only sets its properties. Optionally, a conditions clause defines when a relation can be executed. An example of a data transformation definition is shown below.

```java
transformation Blue2Moon {
  relation POR2LineItemType (lineItem: LineItem, por: Pip3A4PurchaseOrderRequest, index:int) {
    from por {
      orderId = orderId;
      articleId = purchaseOrder/productLineItem[index]/productIdentification/globalProductIdentifier;
      quantity = purchaseOrder/productLineItem[index]/orderQuantity/requestedQuantity/productQuantity;
    }
    create lineItem {
      orderId = orderId;
      articleId = item/articleId;
      quantity = item/quantity;
    }
  }
}
```

Once all relations are defined, we automatically generate a Java class Mapping that implements the actual data transformations. For that purpose, we use OpenArchitectureWare [8]. For example, the relation between operations receiveRequest and search has been implemented by the method por2search() as described in the text box associated with operation search. This method gets as argument the value of behavior item (variable) Pip3A4PurchaseOrderRequest por. This value is assigned after operation receiveRequest has received the purchase order request from Blue.
Step 4: Validation of the mediator PIM. In this step, the design of the Mediator is validated by means of (i) *assessment of the interoperability* between the services of Blue, the Mediator and Moon, and (ii) *simulation* of the interacting behavior of these services. The interoperability assessment method has been presented in [9]. In short, the method checks whether each individual interaction can establish a result and whether the service composition as a whole can establish a result.

The simulation of behaviors is supported by the Grizzle tool [4]. Simulation allows a designer to analyze the possible orderings of operations occurrences, as well as the information results that are established in these operations. In addition, the simulator provides hooks in the simulation process to execute application code upon execution of an operation. This enables us to perform real web service invocations and incorporate the results that are returned by web services during the simulation. For this purpose, stub-code is linked to a modeled web-service operation *call*. This code is generated automatically based on stereotype information that has been retained during the WSDL import, such as the web service’s end-point address and port type name. Furthermore, the simulator allows external web-clients to invoke a modeled web-service operation *execution*. A web service proxy is automatically generated and deployed in an application server, using aforementioned stereotype information. This proxy is responsible for handling the reception of the invocation request and the return of its result. In between, the proxy delegates the calculation of the invocation result to the simulator, which indicates to the user that the operation is enabled and waits till the user requests the simulation of this operation.

The support for real, also called ‘live’, web service invocations, allows one to use the simulator as an orchestration engine in which an orchestration can be executed by simulating its ISDL model. This means that that the simulator provides, in principle, an implementation for the Mediator. However, this simulator does not support important properties of an execution environment, such as performance, monitoring, etc. Therefore, in the next step we transform the Mediator design to a BPEL process.

Step 5: Derivation of the mediator PSM. In this final step the service PIM of the mediator is transformed into a PSM. In our approach, we do not assume a particular execution platform. For example, the service PIM can be transformed to a WS-BPEL specification, EJB, or .Net application. In this section, we present an abstract architecture of possible execution platforms. Fig. 6 depicts this architecture.

![Fig. 6. The architecture of the abstract execution platform](image-url)
The abstract architecture of the Mediator consists of two main components: a Control Flow Manager and a Data Flow Manager. The Control Flow Manager is responsible for sending and receiving messages in a particular order as well as for querying and updating the state of the Mediator. The Data Flow Manager in turn, is responsible for managing the state of the Mediator and for performing the necessary data transformations and constraint checking.

The Control Flow Manager consists of three subcomponents: a Message receiver, a Message sender and a Coordinator. The Message receiver is responsible for receiving all inbound messages and the Message sender for sending all outbound messages. The Coordinator executes the behavior specified in the behavioral model of the Mediator, i.e., based on the current state it activates and deactivates the Message receiver and Message sender. When a message is received, the Coordinator interacts with the Data Flow Manager to update the state of the Mediator. When a message is to be sent, the Coordinator interacts with the Data Flow Manager to obtain the data required to construct the outbound message.

To derive the Control Flow Manager we adopt and extend the approach described in [2]. Our transformation is divided into three successive steps: pattern recognition, activity replacement and model realization. Fig. 7 depicts these steps.

![Fig. 7. Transforming the service PIM of the mediator to a service PSM](image)

The first step recognizes the control flows in a service PIM and then transforms the service PIM to a pattern-oriented service model in the Common Behavioral Patterns Language (CBPL). Each CBPL pattern represents a control flow that is common to most execution languages, i.e., sequence, concurrence, selection and iteration. A sequence contains one or more activities to be executed in succession. A concurrence contains two or more activities that can be executed independently. A selection contains one or more cases to be selected, where a case contains an activity to be executed when its condition holds. An iteration contains an activity to be executed repeatedly as long as its condition holds.

The second step replaces data transformations and constraint checking in the pattern-oriented service model with operations for interacting with the Data Flow Manager. This steps results in a control-flow service model that represents the Control Flow Manager in CBPL.

The last step maps the control-flow service model onto a service PSM. A service PSM contains information that is not present in the service PIM. Examples of such information are the XML namespaces of the exchanged messages or the WSDL port types and operations of the services to be integrated. To provide the required platform-specific information we annotate the elements of the service PIM. This information is maintained during the first and second steps and is used in the last step.

The Data Flow Manager consists of two components: a State manager and an optional Reasoner. The State manager is responsible for updating the state of the Mediator (after receiving a message) and for querying that state (before sending a
message or when checking a constraint). In some cases, data in the received message may have to be transformed before updating the state. For that purpose, the State manager uses the Data transformer component. Likewise, in some cases the State manager uses the Data transformer to construct new messages. The Data transformer is in fact the component that implements the mapping relations specified in the information model of the Mediator. The Constraint checker queries the state of the mediator and determines whether a constraint holds or not.

To take full advantage of the formal specification of the information model of the Mediator, the Data Flow Manager may contain a Reasoner component. The Reasoner uses the formal knowledge specified in the information model of the Mediator in conjunction with the facts about the current state of the Mediator to infer new state information, i.e., it makes all implicit knowledge about the state more explicit. In addition, the Reasoner can be used by the Data transformer and the Constraint checker as an intelligent query engine and constraint solver.

In our solution, we use the ActiveBPEL engine to realize the Control Flow Manager and an external web service to realize the Data Flow Manager. The ActiveBPEL engine executes the WS-BPEL specification generated from the PIM model of the Mediator. The web service stores and retrieves data, exchanged in the messages between Blue, Moon and the Mediator, and performs the data transformations defined in Step 3.

4 Related approaches

Several approaches and solutions have been proposed within the SWS Challenge. Here we briefly discuss the approaches reported at the past edition of the workshop held in Tenerife, Spain. The proposed approaches were based on the WSMO, jABC and FOKUS frameworks. For a more detailed comparison please refer to [6].

The DERI approach [12] follows the Web Services Modelling Ontology (WSMO) framework. It consists of four main components – ontologies, goals, web services and mediators. Data mediation is achieved through the design and implementation of adapters specifying mapping rules between ontologies. During runtime, the approach considers specific mediator services which perform data transformations at entity instance level. The mediator interaction behavior is described by means of Abstract State Machines, consisting of states and guarded transitions. A state is described within an ontology and the guarded transitions are used to express changes of states by means of transition rules. However, this implicit behavior specification may be neither intuitive nor trivial to make sure that the expectations implied by the designed transition rules match the expected operation message exchange patterns.

The jABC solution [13] uses SLGs (Service Logic Graphs) as choreography models, allowing the designer to model the mediator in a graphical high level modeling language by combining reusable building blocks into (flow-)graph structures. These basic building blocks are called SIBs (Service Independent Building Blocks) and have one or more edges (branches), which depend on the different outcomes of the execution of the functionality represented by the SIB. The provided model driven design tools allow the modeling of the mediator in a graphical high level modeling language and support the derivation of an executable mediator from
these models. More recently [7], the approach has focused on how to apply a tableau-based software composition technique to automatically generate the mediator’s interaction behavior. This uses a LTL (Linear Time Logic) planning algorithm originally embedded in the jABC platform. However, the applicability of automated synthesis of the mediator’s business logic is still limited considering the kind of assumptions being made. In comparison with the jABC approach, the approach presented in this paper does not cover automated synthesis of the mediator logic as it intentionally leaves the planning task to the business domain expert.

The core concept of the FOKUS [1] approach is the integration of ontology mappings into BPEL processes. The approach addresses the data mediation by applying semantic bridges to mediate between different information models and representations. Semantic bridges are described as a set of description logic-based axioms relating entities in business information models that are defined in different ontologies but have a similar meaning. The description logic-based data model provided by ontologies in conjunction with semantic bridges allows for applying automatic semantic matching and reasoning mechanisms based on polymorph representations of service parameters. The interaction behavior of the mediator has been manually designed and addressed by using a BPEL engine as the coordinating entity. Some BPEL enhancements were developed to integrate semantic bridges and to support data flow specifications in terms of rules. These enhancements were implemented as external functions that can be plugged into BPEL engines. Thus, in contrast to our approach, the presented approach designs the mediation solution at technology level. It relies strongly on the existing Web standard BPEL and cannot easily be used with alternative technologies.

5 Conclusions and future work

In this paper, we presented a model-driven method for the semantic integration of service-oriented applications. The key feature of the proposed method is that semantically enriched service models are employed at different levels of abstraction to develop end-to-end integration solutions from business requirements to software realization. Therefore, the integration problem is solved at a higher level of abstraction by business domain experts and then (semi-)automatically transformed to a software solution by adding technical details by the IT experts. This way, the same business integration solution can be reused to implement different IT integration solutions using different implementation technologies. In addition, our framework provides a means to define domain-specific languages (DSLs). This way, business domain experts can solve integration problems using concepts that are closer to their domain, thereby, abstracting from complex service representation techniques and the syntax of data transformation definitions.

Currently, we focus on techniques to automate parts of the composition process of the Mediator. In particular, we consider backward-chaining techniques to discover causal relations among the activities performed by the mediator. In our approach, we start with the activities that send messages and recursively search for activities that provide the information required to construct these messages. The search is performed using the mappings defined in the information model of the mediator. In order to
support semantic matching in this search, we use OWL to express the information model and facilitate reasoning about the mappings.

References

15. UDEF. http://www.opengroup.org/udefinfo/.
Semantic Web Services Challenge 2008

Advances in Solving the Mediator Scenario with jABC and jABC/GEM

Jens Lemcke, Matthias Kaiser, Christian Kubczak, Tiziana Margaria, Björn Knuth

1 SAP Research, CEC Karlsruhe, SAP AG, Karlsruhe, Germany, jens.lemcke@sap.com
2 SAP Res. Center Palo Alto, SAP Labs LCC, Palo Alto, USA, matthias.kaiser@sap.com
3 Chair of Software Engineering, TU Dortmund, Germany, christian.kubczak@cs.uni-dortmund.de
4 Chair of Service and Software Engineering, Universität Potsdam, Germany, margaria@cs.uni-potsdam.de

Abstract. This paper extends the previously presented solutions to the Mediation Scenario to cover the advanced scenarios “Purchase Order Mediation v2” and “Payment Problem”. Here, we use first jABC’s technique for modeling service orchestrations via SLGs to solve the scenarios, and then we reuse part of that work and jABC’s capabilities to provide an alternative solution with GEM, a goal-driven approach of configurable, communicating agents that each act upon a responsible human user. In the initial implementation presented at the previous SWS Challenge workshop (June 2008, Tenerife), we specifically implemented those parts that were needed to tackle the “Purchase Order Mediation” scenario. This paper completes the implementation of our vision to handle all three mediation scenarios.

1 Introduction

The purchase order mediation scenario (short: “mediation 1”) has been solved in [1] in a systematic, model driven, service oriented way with the jABC environment, [2]; and in [1] we showed how to substitute the manual design of the mediator’s orchestration with an automatic workflow generation technique that uses declarative user knowledge expressed in terms of semantically enriched LTL (linear time logic) formulas. In [3] we showed an alternative to the LTL-based generation that combines the goal-oriented enterprise management (GEM) approach [4] with the jABC environment, reusing part of the pure jABC solution.

This paper continues the work presented there by addressing the more demanding purchase order mediation v2 scenario (short: “mediation 2”) using jABC+GEM. The payment problem (short: “mediation 3”) has been solved completely, which will be presented in this paper, using jABC without GEM. During this endeavor, Web service connectors and graph segments necessary to solve mediation 3 were created in jABC. For the workshop in October, we plan to reuse these components to solve mediation 3 using the dynamic approach of jABC+GEM in addition to the static approach. The
contribution of this paper is a description of how our implementation for mediation 1 could be adapted to solve mediation 2 and 3.

The rest of this paper is structured as follows. Section 2 summarizes the newly addressed mediation scenarios. Section 3 and 4 present a solution to mediation 3 solely using jABC. Section 5 and 6 present GEM and its modeling style, applied to the Mediator 2 problem, and Section 7 discusses the concrete solution, its architecture, and the ease of adaptation to the new scenario. Finally, Section 8 draws our conclusions.

2 The Mediation Scenarios

This section summarizes the changes of mediation 2 and 3 over mediation 1. A detailed description of each scenario can be found on the semantic Web services challenge (SWSC) Web site.¹

Mediation 1 and 2 consist of two roles: customer, the company Blue, and a manufacturer, the company Moon. The customer triggers a purchasing process by initially sending a purchase order request message and expecting a purchase order confirmation from Moon. This is the requested interface. Company Moon uses a customer relationship management (CRM) and an order management system. Both systems’ interfaces are given as a set of Web services that have to be mediated in order to match the requested interface of Blue.

**Mediation 1** Here, the necessary service calls to and the answers from the provided interface can be determined from a concrete purchase order request. First, an internal customer identification needs to be obtained. Second, a new order needs to be created. Third, a line item needs to be added to the new order for each line item in the purchase order request. Fourth, the order needs to be closed. Fifth, a confirmation has to be received for each of the line items added to the order. And sixth, the purchase order confirmation has to be sent back to Blue.

We showed previously how mediation can be modeled elegantly as a Service Logic Graph in the jABC, or generated by - alternatively - a single call to the LTL algorithm or to the event calculus planner at the core of the GEM approach.

**Mediation 2** differs in that the necessary service calls cannot directly be determined from the concrete purchase order request. This is because the type of confirmation received for a line item added to the order determines the further mediation. If a line item was accepted, the mediation equals the one in mediation 1. Otherwise, if a line item was rejected, a production management system needs to be informed of the line item. The service responses of the production management system are again deterministic, and the rest of the mediation equals the one in mediation 1.

The solution of mediation 2 is simple: it has been easily implemented in jABC without any change to the underlying modeling approach, but it requires multiple calls of the planners, due to the change of knowledge during business process execution.

**Mediation 3** is a payment authentication process, where again Blue and Moon participate. Blue triggers the process by sending a payment initiation and expecting a payment status message upon completion. This time, Blue provides services that have to

be invoked with the right parameters in the right order with respect to non-deterministic answers of previously invoked services. The correct invocation strategy is given by a set of business rules. In particular, a payment initiation request can be directly sent to Blue’s accounting department system if the amount is below 2000 Euro. Otherwise, the payment initiation request needs to be accompanied by an authorization code that must be obtained from a management department system. The management department system may be asked for a specific authority to provide an authorization code. Each authority may provide an authorization code only for amounts below a specific designated amount. If multiple authorities qualify, the one with the least designated amount must be consulted first. After a denial, the next superior authority may be asked.

In addition to mediation 1 and 2, mediation 3 requires a preference ranking to select a specific one from multiple competing goals to pursue next. From the pure jABC point of view, it is still just another instance coverable with the same modeling approach. For the planning approaches it requires capturing these elements in an adequate way.

In the following section, we describe our static solution to mediation 3 using solely jABC before we continue with the description of the dynamic jABC+GEM system.

3 Basic Web Service Handling and Parameter Grounding Techniques

Dealing with Web services assumes that an appropriate mechanism for import and export of service components is provided by the underlying framework.

3.1 Web Service Import

To work with external Web services, we need to import them into the framework and make them available as building blocks. This is done by the import process shown in Fig. 1 which, in short terms, takes a WSDL location and generates Service Independent Building Blocks from it. This functionality is offered as part of the jETI plugin [5] to the jABC.

The generated SIBs take Java binding objects as parameters representing the input and output messages of the imported Web service. If these objects are of primitive types only (e.g., an integer or a character string), there are direct correspondents inside jABC and the grounding to the ontology layer can be done one-to-one. For structured, custom parameters, like an address field or a purchase order request, the provision of the appropriate data structures has to be done manually. We do it once to Java for the pure jABC solution, and once again to the appropriate representation for GEM.

3.2 Web Service Export

In [6] and in more detail in [1] we presented how in jABC we automatically generate a valid Web service from a given SLG. This capability is necessary to achieve a working communication interface for all mediation scenarios. Fig. 2 shows the jABC solution to mediation scenario 2 modeled as an SLG. It is transformed into a Web service that
Fig. 1. The Web service import process inside jABC.

consumes an appropriate input message and it is bound to a representing Java object on the implementation layer.

The specification of valid input and output parameters is done by the application expert who models the workflow. Manual grounding happens when writing jABC services (called SIBs) that consume the input message. In Fig. 2 the Start SIB takes Blue’s input message to the mediator, and grounds it for the jABC to its representing Java object and the ExtractSearchCustomer service extracts from it the information needed for the mediation.

4 Solving Mediation 3 as jABC Service Assembly

We solved now also the payment scenario using the jABC modeling approach described in [1] for the mediation scenarios 1 and 2. We did not expect any particular challenge, so we proceeded by applying the standard jABC methodology already successful with the previous two scenarios, and viewed this case study as an occasion to assess our established methodology for this class of problems. In fact, while the Mediation 2 builds on Mediation 1 and extends it, Mediation 3 is a technically independent problem, that uses an entirely new set of services.

We followed the methodology described in [1,6], that proceeds in three phases: Web service import, orchestration design, and execution, with (optional) Web service export of the solution.
Web service import. We first imported all the Web services provided by the SWS organizers for their use as SIBs inside the jABC as described in Sect. 3.1.

Orchestration of the mediator. Afterwards, we simply orchestrated our solution using these Web service SIBs as well as some customized SIBs that read and write data to and from the challenge’s online services and evaluate the service’s responses. Fig. 3 shows the Service Logic Graph of the payment scenario solution, as it looks inside the jABC. In this SLG,

- Initially we receive a Payment Initiation call from the Accounting Department which is processed by ReceivePI.
- Afterwards, we retrieve Moon’s bank account data from the Financial Information Provider invoking the generic FIPService_GetBankingDataSIB component.
- Now that we have all necessary information, we can start invoking the payment using FDService_ProcessPaymentSIB. The service’s response is then checked using CheckPaymentInitiationResponse.
- If payment was accepted by the back end system, we simply initiate the transaction calling PaymentService_InitiatePaymentSIB.
- Otherwise, we need legitimation from the the Payment Authorization component. To achieve this we extract the appropriate authority data from the response we got when calling FDService_ProcessPaymentSIB.
- The authorization request is then stated invoking MDService_Authorize.
- Afterwards, the response is evaluated using CheckAuthorizationResponse.
- If the payment was authorized, we continue by calling FDService_ProcessPaymentSIB, which should now lead to valid Payment Initiation response in the next step.
- If the payment was not authorized, we check all valid authorities from the response’s data set until either we get an approved response or there are no more authorities to inquire for authorization.
In the latter case, we have to initiate a non valid payment calling PaymentService_InitiatePaymentSIB with a failed authorization.

To read out the status response, we use ReturnStatusFromContext in the last step, leading to either a successful response or to a rejection of the whole payment.

**Execution and Web service export.** The resulting service assembly can then be executed directly within the jABC via the Tracer, our SLG interpreter, or be exported as stand-alone Web service with the technology mentioned in Sect. 3.2.

**Preparation for jABC/GEM.** This solution provided us at the same time with the building blocks needed to apply the combined jABC/GEM technology to this problem. At the time of writing, the combined jABC/GEM solution is not yet completely implemented.

We were however able to complete the implementation of Mediation 2 with jABC/GEM. The combined approach has already been presented in [7], and will not be repeated in depth here. In order to understand and appreciate the difference, we explain here briefly its central traits.

## 5 How GEM sees the world

Our goal-oriented enterprise management (GEM) approach takes a high-level view on the enterprise it manages. Therefore, GEM does not deal with technical details such as concrete Web service invocation and data mediation. These are indeed relevant topics, but in order to simplify and automate enterprise management as much as possible, GEM rather takes a conceptual or ontological view basing on top of the technical view implemented by the basic jABC. Similarly to the LTL based solution for the automated
generation of workflows in jABC presented in [8], GEM is consequently more interested
- in “what” is given and “what” is to be done,
- but not in “how” things are given or done.

In order to be operable, the basic jABC’s technical and GEM’s ontological view have to be interlinked. The link between jABC and GEM consists of communicating
- operation invocations and
- operation responses.

We are going to elaborate on both with respect to the example of invoking the operation and recognizing the response of the “Confirm/Refuse Line Item” operation of mediation 2.

5.1 Operation invocation

The execution of a plan in GEM inherently involves the invocation of operations. For example, \( n \) addLineItem operations have to be invoked, before a close order request can be submitted, and afterwards \( n \) line item responses have to be processed.

But in contrast to knowing “how” the exact Web service operation signature looks like, GEM needs knowledge of “what” an operation invocation requires in terms of the status of the world. Consequently, GEM has a
- view of the world and
- knowledge of operation preconditions.

The general model and background has been already described in [3], therefore we concentrate here on the concrete modeling aspects relevant for this case study.

The world in GEM. GEM’s ontological view of the world is established by the situation, expressed as a file of event calculus fluents that describe GEM’s perception of the world. For example, before invoking addLineItem, the situation could look as depicted in fig. 4.

\[
\begin{align*}
&\text{por(por1),} & \text{por_custid(por1,custID1),} \\
&\text{por_item(por1,porli1),} & \text{porli_prodid(porli1,prod1),} \\
&\text{por_item(por2,porli2),} & \text{porli_prodid(porli2,prod2),} \\
&\text{po(po1),} & \text{po_custid(po1,custID1)}
\end{align*}
\]

Fig. 4. Situation before addLineItem.

The situation description states that currently, there is a purchase order request —por1— belonging to customer identifier —custID1—. The purchase order request
contains two line items —porli1— and —porli2—. Each line item has an attached product, that is —prod1— and —prod2—, respectively. In addition, there is a purchase order object —po1— of the same customer —custID1—.

We notice that this is the situation that should hold directly before invoking addLineItem in mediation 2. A customer identifier has been found for the customer in the purchase order request, and a respective purchase order object was already created for the purchase order request.

The event calculus fluents that make up GEM’s view of the world correspond to XML structures of real Web services. However, the fluents are an abstraction in that

– they only contain as much detail as needed for GEM to perform its tasks and
– they comprise an aligned view of the potentially heterogeneous data type definitions the actual Web services use.

Operation semantics in GEM. Now that we know the current state of the world, let’s have a look at the precondition of the addLineItem operation in fig. 5.

<table>
<thead>
<tr>
<th>Operation—addLineItem([PO,PORLI])—</th>
</tr>
</thead>
<tbody>
<tr>
<td>por(POR), por_custid(POR,CUSTID),</td>
</tr>
<tr>
<td>por_item(POR,PORLI), porli_prodid(PORLI,PRODID),</td>
</tr>
<tr>
<td>po(PO), po_custid(PO,CUSTID)</td>
</tr>
</tbody>
</table>

Fig. 5. Precondition for addLineItem.

In contrast to the state of the world, the precondition now contains variables. An identifier starting with a capital letter denotes a variable. The precondition states that

if there is

– some purchase order request (variable —POR—) for a customer —CUSTID—,
– some line item —PORLI— of that purchase order request (—POR—), and
– already some purchase order —PO— for the same customer as the purchase order request (—CUSTID—),

then addLineItem can be invoked

– for purchase order —PO— with
– the purchase order request line item —PORLI—.

The first list in brackets in fig. 5 —[PO,PORLI]— names the input parameters that addLineItem takes still in terms of the ontological view GEM has of the world. When invoking addLineItem, GEM tells jABC to invoke addLineItem with the instances —po1— and —porli1— of the parameters that GEM has detected in the situation. Actually, the situation allows GEM to call addLineItem twice, one with the parameters
jABC performs a mapping of the ontological object identifiers —po1—, —porli1—, and —porli2— back to the XML structures of the purchase order and the purchase order request line items that it once received from the “Receive PO endpoint” and create-NewOrder operation.

5.2 Operation response

In addition to triggering operations, GEM must be able to react to operation responses. GEM’s communication paradigm is strictly asynchronous. That means that upon triggering an operation, GEM does not suspend. Instead, GEM continues to observe the situation and act upon interesting patterns as prescribed in the plan a GEM agent executes. Only when the plan foresees to explicitly wait for an operation’s response, GEM suspends the current plan until the response was detected. We explain this in more detail on the “Confirm/Refuse Line Item” operation.

The “Confirm/Refuse Line Item” operation is interesting because it can yield different outputs in mediation 2. Depending on the output, different follow-up actions have to be taken. Upon a confirmation, the item can be directly put to the purchase order confirmation. Upon refusal, the item must undergo the process of a production request.

The situation before receiving any response from “Confirm/Refuse Line Item” may look as depicted in fig. 6. In addition to the situation depicted in fig. 4, now already the result of the addLineItem operations is reflected in the situation. In particular, GEM knows that now the purchase order —po1— contains the two purchase order line items —poli1— and —poli2—. Please note the difference between purchase order request line items (—porli(...)—) and purchase order line items (—poli(...)—).

The GEM system can react to either of the responses by just observing the situation. The situation is updated by jABC. jABC contains the actual Web service endpoint that waits for the response from “Confirm/Refuse Line Item”. Let’s consider that the first line item is confirmed and the second gets refused. Upon receipt of each response, jABC updates the situation by adding statement to the situation. After both responses, the situation looks as depicted in fig. 7.
Further actions of GEM can now react to the new situation. In our example, a line item agent would start to kick off a production request workflow for —porli2—, which was refused. The proceeding is analogous to section 5.1.

6 Separating GEM to multiple gents

As GEM internally utilizes AI planning, it has to respond to the common critiques of how the many declarations in an AI system may be efficiently maintained and how the expensive planning computations at run time can become feasible. Our answer is the separation of the GEM system to multiple agents, called gents. Conceptually, a gent is responsible for a business domain, such as the CRM, stock management, or production management in mediation 2.

As gents partition the enterprise, also operations, scope, and purpose specifications are separated to the individual gents. A multi-gent GEM system therefore consists of

- one situation file commonly read and written by all gents,
- a generic implementation consisting of the five procedures explained in [3],
- a configuration per gent, consisting of a set of operations, a set of scoping patterns, and a set of purpose rules, and
- a set of files per gent storing run-time data of that gent.

All communication between gents within GEM and the notice of messages received from external services is exclusively performed through the situation. Conceptually, a gent reads from the situation and invokes operations. The result of an operation is expected to show up in the situation so that any agent may act upon that result.

Through partitioning the configuration, GEM makes an effort to make planning more feasible and to increase maintainability. Feasible performance of the planning is facilitated as the planning component has to observe the restricted set of operations that is local to the current gent, only, in contrast to all operations available in an enterprise system. As for maintenance, configuration is restricted to smaller business units instead of handling the large configuration conglomerate of the whole enterprise system. In addition, however, some more high-level maintenance needs to ensure a proper cooperation of the gents in a GEM system. We feel that this approach naturally reflects
current practice in enterprise management where it is common to structure an enterprise according to organizational units and reporting lines.

The concept of cooperation across business units should be exemplified based on mediation 2.

**Gents for Mediation 2** Our GEM system for mediation 2 consists of the three gents: the CRM, lineitem, and sales gent.

The CRM agent is configured to react to a purchase order request without a customer ID. Its only operation is the retrieve customer-ID operation which is capable of fulfilling the goal of generating a purchase order request containing a customer ID.

The sales gent is configured to react to a purchase order request with customer ID. Its goal is to generate a purchase order confirmation. To fulfill this goal, the sales agent utilizes the create order, add line item, and close order operations. In addition, the sales agent waits for a line item confirmation. Following our approach, an incoming line item confirmation appears in the situation. If the appearing confirmation contains a rejection, the lineitem agent is triggered as this pattern is defined as its scope.

The lineitem agent checks the production capabilities and orders the production of the product. The production request in the situation lets the sales agent complete its execution.

7 **Grounding GEM to Web services**

In order to solve mediation 2 we created the three agents just described: the CRM, the sales, and the lineitem agent.

The Service Logic Graph (SLG) which realizes in jABC the mediation workflow is created as explained in [3] with the set of agents and their specific operations. Fig. 8 shows the repeated use of the known Graph SIB for the agent and operation graphs, to encapsulate the underlying SLG in one SIB. Every agent consists of the same procedures. The agent is published itself as a Web service, and it is called by the agent graph in jABC.

![Fig. 8. The Mediation workflow in GEM/jABC.](image-url)
Each operation SIB hides the graph of the service calls specific to each agent, whereby multiple services can be assigned to one agent. Because GEM can schedule the concurrent invocation of multiple services, jABC is able to deal with an unknown number of service calls in one message from an agent.

The SLG for a service call always matches the pattern shown concretely in Fig. 9 for the operation of the CRM agent. It consists of four kinds of SIBs. The most important is the SIB for the Web service offered by the challenge, which is semi-automatically created (import phase). Here we reuse the SIBs already created for the plain jABC solution. In some cases additional mediation SIBs are needed to prepare the data for the Web services. They are reused as well form the plain jABC solution. After every service is executed, a SynchronousResponse SIB is needed to inform GEM about the result of the service. These SIBs are surrounded with the SIBs of the External Context, where initially the necessary data is loaded and finally the result of the Web service is saved.

The External Context stores the real world data for the agents, which are published themselves as Web services. GEM and jABC exchange only the IDs of the real world objects, so GEM does not know their specific structure. The mediation between both data worlds is realized in the SIBs of the external context, which are automatically created from a configuration file.

![Fig. 9. Typical graph for a Web service call.](image)

### 7.1 The global jABC/GEM architecture

Considering the necessary components for every GEM implemented solution (GEM-Server, External Context Server, jABC model), every scenario can extend the architecture with additional servers. The mediation scenario 2 demands the existence of two special services, like in the first mediation.

The concrete overall architecture is shown in fig. 10: The service POR Receiver takes the purchase order request from the challenge testbed and stores the relevant data to the GEM situation and to the External Context. The second service, Line Item Confirmation, receives the confirmations of every line item from the testbed and informs GEM about the receipt.

### 7.2 Evolving jABC/GEM to Mediation 2

The second mediation scenario introduces the Product Management System (PMS), with two services that have to be called if the line item was not confirmed by Stock
Management System before. For that purpose, GEM is extended by a new agent, the line item agent, with two operations corresponding to the PMS services.

Furthermore, the operation which waits for the confirmation of the line item had to be changed. If the line item is refused, another effect is written, so that the planner can schedule the new agent into the plan.

Since GEM requests the PMS service invocation from jABC, the two additional SIBs are needed which call the real Web service of PMS. These are the same SIBs already (semi-automatically) created for the plain jABC solution of the same scenario, so there was no specific effort here.

To deal with the new services, the External Context must be aware of the new data types. The SIBs which load and save the data from and to the context must consider the two services. So, the configuration file was completed with the new types and services with their input and output parameters and types. Afterwards, the DataLoaderSIB and DataSaverSIB were generated, and the External Context was generated and published. These steps are simple and largely automated by now.

Finally, the models in the jABC had to be updated to include the new agent and services. The line item agent in the top model, see fig. 8, was created by using the known agents GraphSIB with the agent key (from GEM). Two models for the operations were created following the pattern from fig. 9 using the newly created SIBs for the Web services calls. Additional SIBs for data mediation were not necessary. The two models for service calls were connected with the OperationSwitchSIB in another model, which is encapsulated in the top model by the the GraphSIB “LineItemAgent Operations”.

We can conclude that adapting our jABC/GEM solution to the first mediation scenario presented in [3] to the new mediation 2 could be done with a limited effort. The main changes on the GEM side could completely be done in the configuration of the agents. The main changes on the jABC side involved the reuse of existing data mediators and the semi-automatic generation of new data mediators. This makes us confident to be able to complete the implementation of mediation 3 using jABC/GEM until the upcoming SWSC workshop in October.
8 Conclusions

In this paper we showed how to extend the previous jABC solution to the Mediation 1 and 2 to the payment scenario and the more declarative jABC/GEM solution of Mediation 1 to the Mediation 2. In particular, we have elaborated on the concrete modeling in GEM of the Challenge’s services and operations, and discussed in detail the changes needed on the structure, configuration, and jABC layer of the solution. We have also been able to assess on an independent problem (mediation 3) the generality of the solution’s methodology in jABC and the degree of automation already achieved. In fact, we did not need to extend or modify anything of the jABC concepts, modeling style, and framework capabilities. Ongoing work concerns the implementation of Mediation 3 with the jABC/GEM approach.

References