SWS-ASE: Leveraging Web Service-based Software Engineering

Ulrich Dinger
TU Dresden & Siemens AG
Hans-Grundig-Str. 25
01062 Dresden, Germany
dinger@rn.inf.tu-dresden.de

Roy Oberhauser
Aalen University
Beethovenstr. 1
73430 Aalen, Germany
roy.oberhauser@htw-aalen.de

Christian Reichel
ETH Zurich & Siemens AG
Universitystr. 6, CAB F57
CH-8092 Zurich
christian.reichel@inf.ethz.ch

***Abstract*—Web Service (WS)-based integration is a broadly accepted technique to address the heterogeneity in current software systems. With regard to the growing complexity of state-of-the-art software engineering (SE) processes, Semantic WS (SWS) can be a strategy to manage the heterogeneous SE software tools and inherent cross-dependencies in an automated way. This paper presents an interoperable tool approach for Automated Software Engineering (ASE) which comprises a distributed SE process management framework based on SWS. It includes plug-ins to visualize the generated workflows as well as to manage the feature dependencies of SE choices, e.g., with regard to quality criteria. Application of the SWS-ASE infrastructure within industrial use cases yielded significant improvements in areas such as reliability, usability, and efficiency. Within this paper, the results of such a use case are discussed and depicted in detail.**

Keywords-Software Engineering; Semantic Web Services; Software Automation; Reasoning; Ontology; Feature Modeling; Software Tools; Computer-Aided Software Engineering

I. INTRODUCTION

In support of Service Oriented Architectures (SOAs)/WS and the Semantic Web vision [1], the W3C, OASIS and other organizations such as WSMO and the SWSI, SWSS, DAML group, have developed specifications and approaches. Basic efforts have focused on the automatic discovery and composition of Semantic Web Services (SWS). The use of semantic models for industry domains in combination with semantically annotated web services (e.g., via WSDL-S [2], OWL-S, WSML) makes it possible to derive logical inferences via a reasoning process. For the software engineering (SE) domain, the principles of the Semantic Web vision are applicable and beneficial to various challenges it faces, especially with regard to expressing a subset of SE information in a machine-processable (and thus automatable) form [3].

There is a continuous and growing demand to produce software in shorter cycles with higher productivity and better quality. The heterogeneous toolchains (e.g., MDA and AOP frameworks, build and deployment tools) common in SE create difficulties when viewed holistically from an Automated Software Engineering (ASE) perspective. Typically a large set of explicit and implicit feature choices are made for a software architecture and application during various SE process stages, the effects of which are not always evident. Implicit dependencies elsewhere in the toolchain workflow often affect both SE product and process qualities. The paucity of machine-processable feature classifications spans technology and design choices to ISO 9126 [4] quality attribute tradeoffs. As a tangible example, consider a security mechanism while modeling - should it be implemented/generated, or may runtime support already exist in the eventual application server? Some issues resulting from the manual steps (or manually-created toolchain workflows) between tools are brittleness, hidden assumptions, and time-consuming adaptation. Currently, semantic information about what a tool does, the transformations it performs, what input formats are expected, what output formats are generated, etc. are not easily accessible or machine-processable. A further issue is the lack of standardization, distribution, and efficient asset management in the tool arena (e.g., search, compatibility issues, integration, and update notifications), making tool software assets costly and the process error-prone.

To address this situation, a unifying and comprehensive approach called SWS-ASE is explored that automates the selection and composition of tool functionality. The main ideas are to attach semantic information to existing SE tools, to create a tool registry of distributed and local services, to use reasoning engines for automatic composition and, therewith, to bootstrap Semantic Web Services for the SE process of WS-based Applications. Using such an approach, yields significant benefits by:

1. Integrating heterogeneous, distributed tool functionality in a standardized way
2. Improving feature dependency selection support using a classification system
3. Automating toolchain workflow composition and execution

Although it is unrealistic to expect complete automation under all circumstances, SWS-ASE enhances the engineer's capabilities and provides CASE-like benefits to the SE domain, especially with regard to usability, reliability, and efficiency.

The remainder of this paper is structured as follows: section II analyses an example scenario where state-of-the-art SE tools are used. In section III, the SWS-ASE solution approach is

presented wherein the OWL-based ontology (domain model) is introduced with the typical artifacts, procedures, and tasks for software engineering and deployment. Moreover, the specific parts of the overall SWS-ASE process (such as the analyzer, partitioner, etc.) are described. Section IV evaluates the solution results. Related work is discussed in section V. This is concluded with a summary and a brief outline of future work.

II. MOTIVATING EXAMPLE

In order to elucidate current SE difficulties in the toolchain, this section introduces an example scenario that is engineered using several state-of-the-art technologies and methodologies. The underlying use case was extracted from the TPC-W benchmark [5], which depicts a retail store application where customers can visit a web site, look at products, place an order, request the status of an existing order, etc.

A. BookService Scenario

Fig. 1 shows a simplified sequence diagram for one operation of the main application service (*BookService*) that provides the business logic for an online book reseller.

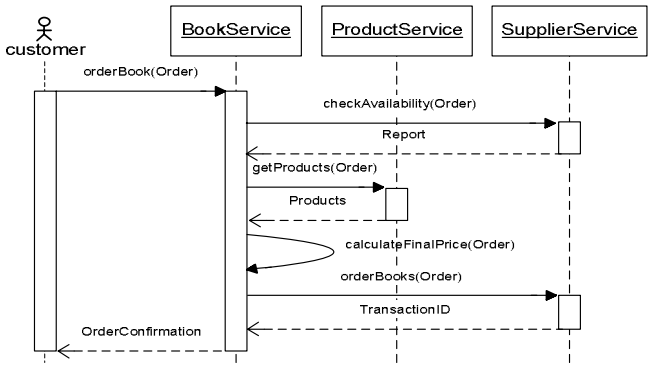

Figure 1. Sequence diagram (orderBook)

In this context, a *customer* is able to call the *orderBook* operation of the *BookService*. The availability of the *Order* is checked via an external Web Service call *checkAvailability* to the *SupplierService*. As a result, the supplier responds with a *Report* containing the availability information. If the products are available, the operation *getProducts* of the *ProductService* is called which sends back the detailed product description. In the next step, the *BookService* calculates the final price of the order. Afterwards, the *SupplierService* is called again to trigger order shipping. In the end, an *OrderConfirmation* with a transaction id is sent to the *Customer*.

B. State-of-the-art SE Process

To engineer the Book Web Service (which integrates the existing Product- and Supplier- Services), various quality attributes must be taken into consideration. A set of decisions made during the SE process impact these quality attributes, for instance the final Quality-of-Service (QoS) provided by the BookService. One possible illustration of typical steps to engineer the BookService involves:

1. Platform-independent modeling via UML (class diagram, state diagram, etc.)
2. Feature decisions, e.g., asynchronous calls, profiling, security such as message encryption, tracing
3. Usage of AOSD tools at development time, e.g., AspectJ, to realize message encryption via static aspect weaving
4. Code and template generation decisions, e.g., EJB as the target technology (using MDSD generators, Eclipse EMF, etc.), asynchronous calls via Message-Driven Beans (MDBs), Web Service enablement
5. Application logic implementation
6. Compilation, perhaps based on profiling parameters
7. Packaging tool tasks dependent on the target execution environment and version, e.g., a J2EE application server such as JBoss or a non-J2EE framework such as the Spring Framework
8. Execution of a deployment file via ANT, which will depend on the chosen target, configuration, parameterization for profiling, etc.

As depicted, the state-of-the-art SE process includes several manual sub-processes with limited to no automation between the individual steps. For instance, there is currently no infrastructure available to manage the overall SE process (model transformations, deployment, etc.) with regard to the implicit and associated dependencies that need to be considered in the complex chain of singular SE decisions. As the size of the application grows, so does the number of involved artifacts, which has an exponential effect on the number of dependent relationships and decisions that affect quality. Without appropriate management support, the process becomes unnecessarily error-prone, inefficient and unreliable. For example, the feature choice of asynchronous messaging impacts component types (e.g., MDBs vs. Session Beans) which in turn affects certain QoS parameters such as response times and throughput. Furthermore, the inherent complexity due to the number of combinations makes it difficult to obtain an optimal solution with regard to the matrix of theoretically possible quality aspect permutations. Moreover, because of the manual steps within the SE process, other engineers may not be able to reliably reproduce the subsequent decision chain. This could be detrimental when it is not obvious why special decisions were made or what the consequences of these decisions were, e.g., the QoS impact (such as flexibility, performance) of the static weaving of aspects vs. runtime AOP utilization in an application server. Additionally, as shown in the BookService example, the lack of meta-information constrains sub-process interoperability and therefore directly limits the usability of the SE toolchain. As a result, inter-activity CASE cannot be leveraged which in turn negatively affects the efficiency of the underlying SE production cycle.

III. THE SWS-ASE APPROACH

To address the aforementioned challenges, a comprehensive and unifying approach is required that can automate and leverage semantic information to reliably compose the various elements involved in the SE process.

This section introduces the SWS-ASE approach. First, an optimized model for the SE domain is presented. Then, in section III.B.1, a registry of exemplary services that tackles typical domain tasks is described. The sections III.B.2 to III.B.5 describe the process of information retrieval and execution to solve the complex sets of SE requests. In section III.C, advanced features of the underlying architecture with regard to validation and optimized analysis are depicted.

A. Domain Models

In order to handle the artifacts of the SE domain, a model for the domain must exist. In this context, several formats are available such as the OMG's Model Object Facility (MOF) and the XML Schema standard. Within the semantic web world, the Web Ontology Language (OWL) can be used to describe the entities of the domain (concepts) and the relationships among them in ontologies.

Fig. 2 provides an overview of an OWL-based ontology of SE-specific concepts that serves this paper's purposes with the aforementioned online book reseller scenario. Ongoing work on formal ontologies for SWEBOK [6] and related SE areas is needed. The dark circles with the lighter inner circles represent concepts while the single-color circles represent individuals (instances) of concepts (e.g., the "JBoss" entity is an instance of the "AppServer" concept). As shown, the ontology is currently divided into four sub-ontologies.

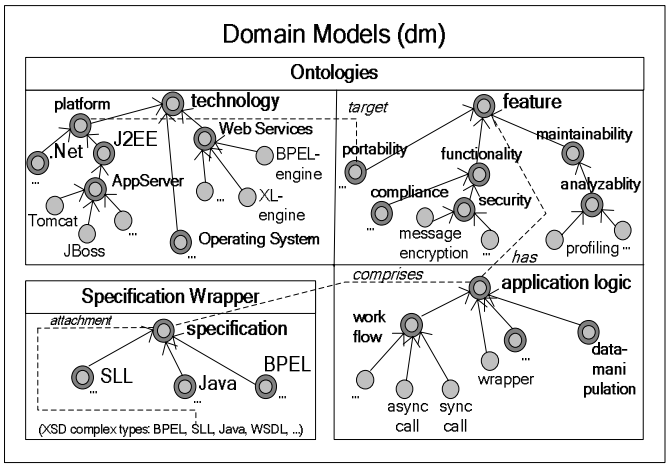

Figure 2. Domain models

The first one (upper left of Fig. 2) represents the sub-domain of target technologies that relate to execution environments. The second ontology (upper right of Fig. 2) introduces the "feature" concept which is able to describe certain characteristics or qualities of a piece of software such as those defined in the ISO 9126 [4] standard. The "Specification Wrapper" ontology on the lower left of Fig. 2 is separated from the other ones because the *specification* concept provides the semantic links (wrappers) for existing programming languages and standards (such as Java, BPEL, and the Service Language Layer (SLL) [7] described later). For each language, a specific sub-concept is created that points to an external XML Schema/DTD or human-readable document that provides a well-defined "external" semantic for the specific concept. The usage of references reduces the mappings needed for the semantic-based conversions between different specification concepts (e.g., from SLL to BPEL or Java). The fourth ontology (lower right of Fig. 3) describes the set concepts that relate to the "application logic" concept, e.g., workflow parts such as asynchronous or synchronous calls. In order to decorate an application logic part (e.g., within a Java document) with specific characteristics, a feature list (e.g., security, profiling, etc.) can be attached. The underlying process (how to analyze and attach the features to specification concepts) is explained in section III.B.4.

B. SWS-ASE Process

Essentially, the complete SWS-ASE process comprises five steps: Service Registration, Knowledge Retrieval, Request Assembling, Workflow Planning, and Workflow Execution. Fig. 3 will be used to illustrate the process. First, the service provider generates and publishes a semantic service description in the *service registry*. A reliability check validates this description, explained in section III.B.4. At a later point in time, a requestor (such as an engineer, deployer) formulates a request (with the help of a software tool) that results in a semantic request that is submitted to the semantic *reasoner*. The *reasoner* then retrieves the information of all known services (using the *service registry*) and plans the workflow(s) that fulfill(s) the semantic request. The user can then submit the planned workflow for execution by a workflow engine and receives the result.

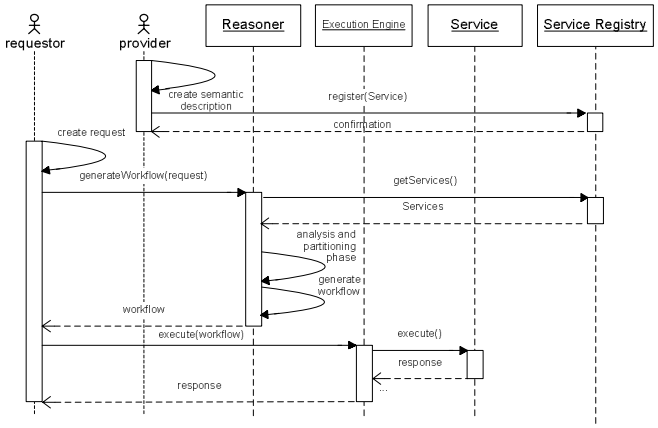

Figure 3. Activities/roles

The following sub-sections will describe each of the aforementioned steps in detail.

1) Service Registration: The implementation of the solution approach is based on a Service-Oriented Architecture (SOA) with Web Services (WS). Thus, all created WS encapsulate a certain piece of functionality that can be reused

within the SE process. Additionally, each service is annotated with semantic information, which describes the pre-/post-conditions (PRC/POC) and effects (E) of a specific service operation. This information is stored in the service registry.

As an example for registered SWS, the AddTracingService of Listing 1 provides semantic information for its addTracing operation using SLL [7], which was developed as a special-purpose, platform independent WS programming language, e.g., to provide better mapping capabilities to different target platforms and technologies. Other formats such as WSML and OWL-S are conceivable. The first six lines import the XML Schema file of the SLL specification, the ontology concepts (for the semantic descriptions), and the semantic rules (RuleML) for the SE domain. As shown on line 11, it takes an SLL document (with the XSD complex type *xsll:unit*) and a configuration document (*tr:configuration*) as input and returns an SLL document (line 12).

Lines 15 to 43 provide a semantic description for the *addTracing* operation. This includes the declaration of the semantic signature (lines 16 and 17) which comprises in- and output concepts. Lines 19-26 contain the mapping between the semantic concept and the XML Schema data types using variables and XQuery/SLL expressions (XSLT could also be used). This is followed by the declaration of pre- and post conditions as well as effects (Lines 28-40).

As shown in line 30, the operation precondition uses a rule from the SE domain that states that the input concept I1 (SLL program) must not have the *op_call_tracing* feature. Additionally, the effect section states that the output O1 (SLL program) inherits all features from the input (line 36) and the feature *op_call_tracing* is added (line 38).

```
01 // register the XML-based SLL model
02 namespace xsll = "http://fxl-project.com/lang/sll/1.1/xsll.xsd";
03 // register the domain ontologies
04 namespace dm = "http://fxl-project.com/ontologies/dm.owl";
05 // register domain specific rules
06 namespace rdm = "http://fxl-project.com/rdm.rml";
07 //…
08 service AddTracingService {
09
10 // this operation adds tracing capabilities to the input sll service
11 public operation addTracing($i1 as xsll:unit, $i2 as tr:configuration)
12    returns ($o1 as xsll:unit) {
13
14    // semantic description
15    semantic {
16      signature ($I1 as dm#sll, $I2 as dm:tracing_config)
17          returns ($O1 as dm#sll);
18
19      mappings {
20       //mapping from semantic to input parameters
21       $i1 = $I1/attachment;  //Note: OWL properties with complex type
22       insert $I2/type into $i2/op_call_tracing;
23       //…
24       //mapping from output parameters to semantic
25       insert $o1 into $O1/attachment;
26      }
27
28      pre_conditions {
29        // prohibit the '#op_call_tracing' feature for the input I1
30        rdm:prohibitFeatures($I1, ['dm#op_call_tracing']).
31        //…
32      }
33
34      effects {
35        // the output O1 inherits all features of the input I1
36        rdm:copyFeatures($I1, $O1).
37        // add '#op_call_tracing' feature
38        rdm:addFeatures($O1, ['dm#op_call_tracing']).
39        //…
40      }
41
42      post_conditions {//n.a.}
43    }
44    // operation body
45    // …
46  }
47 }
```

Listing 1. AddTracingService

Table I shows sample *service registry* entries needed for the BookService scenario. Along with the service name, operation-specific semantic input and output ids and concepts (cp. the semantic signature of listing 1) are provided. The semantic description includes the type as well as the facts/rules. The following abbreviations of domain specific rules are used in the last column: *supportFeatures (sF), prohibitFeatures (pF), hasMinimumOneFeature (hMOF), copyFeature (cF)* and *addFeature (aF)*.

TABLE I. SERVICE REGISTRY ENTRIES

service name	semantic inputs		Semantic outputs		semantic	
	id	concept	id	concept	type	rules/facts
BPEL2SLL	I1	bpel	O1	sll	PRC	sF(I1, [‘workflow’,...])
	I2	config			E	cF (I1, O1)
SLL2EJB	I3	sll	O2	depl_config	E	cF (I3, O3)
	I4	config	O3	xjava*	E	aF (O3, [‘#ebj2.0’])
SLL2WSDLS	I5	sll	O4	wsdls	E	cF (I5, O4)
JavaCompile	I6	xjava*	O5	xclass*	E	cF (I6, O5)
AddTracing	I7	xjava*	O6	xjava*	PRC	pF (I7, [‘#op_call_tracing’])
					E	cF (I7, O6)
					E	aF (I7, [‘#op_call_tracing’])
AddSecurity	I8	sll	O7	sll	PRC	pF (I8, [‘#message_encryption’])
	I9	config			E	cF (I8, O7)
					E	aF (O7, [‘ message_encryption’])
Deployment	I10	depl_config	O8	report	PRC	hMOF (I11, [‘#ejb2.0’,’#WS’,…])
	I11	bundle			PRC	pF (I11, [‘#deployed’])
					E	cF (I10, O8)
					E	aF (O8, [‘ #deployed’])
Bundling	I12	xclass*	O9	bundle	PRC	hF (I12, [‘#ejb2.0’])
	I13	wsdls			PRC	pF (I12, [‘#bundled’])
					PRC	cF I12, O9)
					PRC	aF (O9, [‘#bundled’])

All other services of the service registry are described similarly. In order to avoid dependency on SLL for the semantic service description, the syntactic and semantic parts of SLL can be (and were) translated to WSDL-S [2].

2) Knowledge Retrieval: Fig. 4 depicts the underlying workflow used by the current solution implementation to extract the semantic information needed for the SWS-ASE process. Service Descriptions, including semantic and non-semantic information, serve as inputs to WSDL-S descriptions, which in turn serve as inputs to the Knowledge Base used by the reasoner. Additionally, Domain Facts/Rules including Ontologies and automation Rules also serve as inputs to the Knowledge Base.

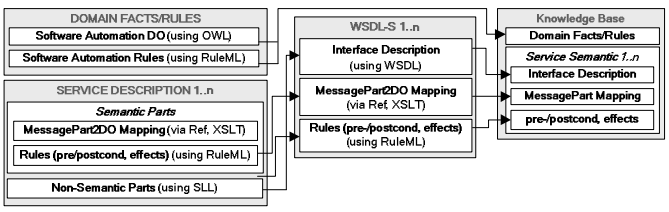

Figure 4. Semantic knowledge extraction

Using the Knowledge Base, a directed services graph $G=(C, E)$ is created, where all I/O concepts of the service operations are represented as a set of vertices $C=\{c_0,...,c_n\}$ and

all connections between the concepts as a set of edges $E=\{e_0,...,e_m\}$ such that $E \subseteq [C]^2$,. In this scenario, a service operation is represented as a sub-graph G'(C', E'), with $G' \subseteq G$, $C' \subseteq C$, $E' \subseteq E$, the input concepts $I_i \in C'$, the output concepts $O_i \in C'$ and the set of directed inner-operation edges $E'(I,O)=\{i_0o_0, i_0o_1, ..., i_0o_k, i_1i_0, i_2i_1, ..., i_ni_{n-1}\}$. However, no real connection between the inputs of an operation exist, thus (ref. to Fig. 5) a requiresConnection (rC) rule is added to all inner-edges of the first operation input (*I3*) in order to model the requirement (transition dependency) that all inputs must be connected. Fig. 5 shows an example Petri net and the corresponding optimized internal graph representation of the SLL2EJB service, whereby the vertex names correlate with the id entries in Table I.

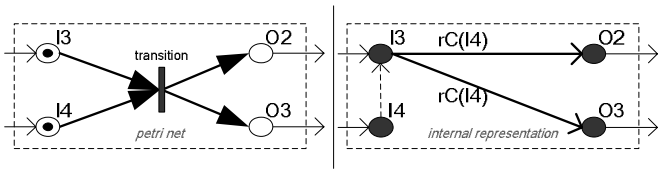

Figure 5. SLL2EJB inner-operation edges

Additionally, two vertices *o* (output) and *i* (input) of two different operation sub-graphs of *G* are connected (adjacent) if their concept types provide a match, meaning *oi* is an edge of G. *oi* is called an outer-operation edge because it connects the output of one service operation with the input of another service operation. Fig. 6 exemplifies how the sub-graph of the SLL2EJB service operation of Fig. 5 can be connected to the concepts of other operations via outer-operation edges. Here the outputs of the BPEL2SLL and AddSecurity services serve as inputs to the SLL2EJB service, whose outputs serve as inputs to the JavaCompile and Deployment services.

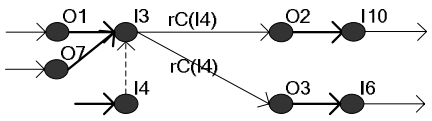

Figure 6. SLL2EJB inner-and outer operation edges

In order to include the rule handling (e.g., dependencies, etc.) in the workflow planning phase, a set of rules $R=\{r1,...,rn\}$ can be assigned to an edge *e*, as shown via the example of the rC rule. Thereby, the function $m:E \rightarrow R$ provides the set of assigned rules of an edge.

3) Request Assembling: In order to automatically convert and deploy the BookService of section II, the requestor formulates the desired objective as a semantic request. In this context, the tuple $Rq=(G, S, T, F)$ represents a request, where $G=(C, E)$ is the directed graph, $S \subseteq C$ the set of start concepts, $T \subseteq C$ the set of target concepts with $T \cap S = \varnothing$, and *F* the set of desired features. In the case of a deployment request for the BookService, the following elements could be assigned:

$S=\{dm\#sll, dm\#config\}$

$T=\{dm\#report\}$

$F=\{dm\#message\_encryption, dm\#profiling,$

$dm\#async\_call, dm\#deployment/target=\#dm:JBoss\}$

The given inputs $S=\{i_0,...,i_m\}$ of the request are the *sll* concept which correlates with the BookService document instance, and a *config* concept which correlates with a directives document instance. In this example, the output *T* contains a *report* concept that provides details of the deployment action. Additionally, the desired features *message_encryption*, *profiling, async_call*, and *deployment* are specified. In order to fulfill the request, a mapping function $Rq \rightarrow W$ is needed which provides a workflow *W* for the given request.

4) Workflow Planning: A workflow (path) is a special, non-empty graph W=(C, E) which can be expressed via its sequence of vertices $W=c_0c_1..c_n$. The set of features $F=\{f_1,...,f_n\}$ which are assigned to a specific workflow *W* can be extracted via the function $f:W \rightarrow F$. In the context of Web service composition, the fundamental problems of planning are addressed by various research efforts, such as AI planning [8] [9] [10]. Solving a planning request can be viewed as a search problem in the space of all possible combinations, where the search algorithm starts with the set of start concepts *S* and goals *T* and *F*. In the domain of SE, two typical scenarios occur that must be covered by an algorithm:

1. The calculation of all workflow alternatives for a specific request.
2. The calculation of an optimal workflow for the request that fulfills a certain criterion (best calculation time, shortest workflow, etc.).

However, both planning problems are NP-hard [11]. To provide better performance, the underlying planning algorithm can use case-specific strategies, including Hill-climbing and heuristic methods. For example, in the current SWS-ASE implementation, the evaluation of the planning state to approximate the distance between the initial states *S* and the target states *T* is realized via an integrated Relaxed Graphplan heuristic [12]. In addition to heuristics, additional information is calculated for each state (e.g., the set of helpful rules *R*). This reduces the branching factor for typical SE scenarios where a fast initial result is needed. Furthermore, specific restrictions (e.g., max. 10 composite services, timeouts) and prohibitions for certain workflow patterns (permutations, sequential execution of independent services) are made. All theoretical investigations on how to improve search and planning algorithms are out of scope for this paper and extensively covered via the efforts within the workflow and AI planning research community (see section V). To illustrate their practical impact, section IV shows promising results and measurements for certain usage scenarios within the SE domain.

5) Workflow Execution: The result of the preceding planning phase is the generated graph W=(C, E) for a specific semantic request, which connects a set of WS from the service registry. Therefore, the graph W can be represented as a WS. For the purpose of workflow description, several languages such as BPEL or SLL [7] are available. One generated workflow for the aforementioned *BookService* deployment request is illustrated in Fig. 7.

Assuming the BookService was modeled in UML and converted to SLL, the SLL document together with a configuration document serve as input *S* to the workflow (upper left boxes) where the output *T* is shown as a report box. In the first step, the *AddSecurityService* is called in order to weave in the *message_encryption* feature (as an aspect) in the SLL program. The *SLL2WSDLService* generates a WSDL-S description from the SLL service input. In parallel, the SLL program is transformed into an EJB that comprises a set of Java files. Then the set of Java files is compiled via the *JavaCompileService*. The ProfilingService instruments the class files to enable the requested profiling capabilities. All class files along with the WSDL-S are bundled via the *BundlingService*. In the last step, the EJB bundle is deployed and registered using the *DeploymentService*. The output of the last service is a concept called *report* containing information about the deployment action.

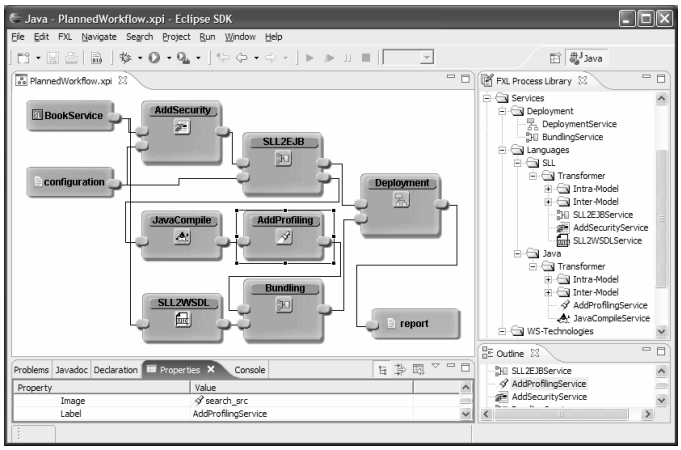

Figure 7. BookService deployment workflow

As previously mentioned, the generated workflow can be represented in various forms, such as SLL or BPEL, and executed on the specific interpreter, for instance, the XL-platform [13] in the case of SLL or a BPEL engine.

C. Feature Analysis Process

A critical issue within the SE process relates to the extraction and creation of meta-information. In the aforementioned BookService application scenario, the requestor manually adds meta-information to the input documents, which is error-prone. For instance, it is not possible in the SWS-ASE process to automatically check whether the *BookService* input document already contains a special feature (e.g., message encryption) or not. Consequently, undetected features could cause errors in the SWS-ASE process, or at least needless overhead for the workflow execution phase when unnecessary services are included (e.g., the AddSecurityService).

To address this issue, an additional *analyzer* entity is included. The task of the analyzer entity is to scan an input document *D* (e.g., the BookService.sll document) of a certain concept type and to detect specific feature patterns *P*. It provides a mapping $a:(P, D) \rightarrow F$. The list of detected features $\{f_1,...,f_n\}$ can then be used in the sub-sequent planning phase.

Fig. 8 shows the analysis result of the BookService.sll document.

As shown, several features such as *wrapper*, *workflow, message encryption,* and *data manipulation* are detected. The current analyzer implementation performs pattern matching on the XML language representations [7] via XSLT by using its template matching functionality with predefined patterns to create a list of semantic features that are then attached to the analyzed document and used within the planning phase.

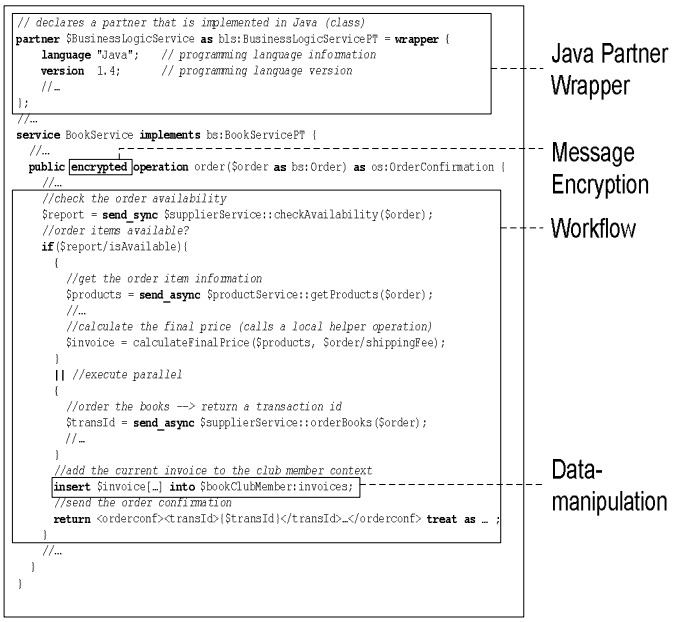

Figure 8. BookService analysis result

Another issue that is dependent on the analysis results relates to the complexity of a specific document. In this context, the SLL programming language can provide an example: as a full WS programming language, it is more powerful than the BPEL language, which can only describe the workflow between Web Services. Therefore, only limited parts of a complete SLL document can be mapped to a BPEL document (external service invocations, control flow, etc.) and others cannot (trigger and monitor clauses, explicit security handling, semantic descriptions, etc). If no possibility to transform a complete SLL program to another target language exists, a promising strategy is to partition the SLL program into code blocks attached with a semantic description and "glue code" that combines these partitions. Fig. 9 shows the *analyzer* in combination with the *partitioner* entity that realizes the partitioning of documents. Basically, a partitioning function $p:D \rightarrow A$ exists which maps a document *D* with a set of features $F=\{f_1,...,f_n\}$ to a set of partitions $A=\{a_1,...,a_n\}$, whereby each partition has at least one attached feature $f \in F$ of *D*. Additionally, the current implementation accepts a set of strategies (splitting rules) as input, in order to define the most appropriate size of a partition.

In this way, the partitioning of an input document (e.g., SLL) enables a larger set of possible service combinations (e.g., pieces that cannot be handled by BPEL alone can be delegated to other technologies).

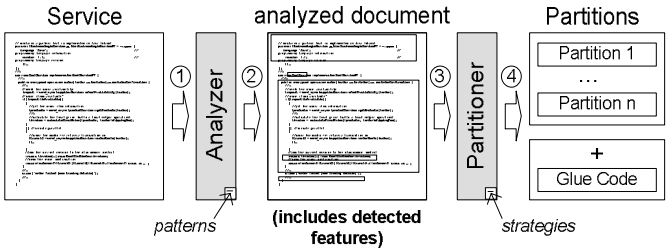

Figure 9. Analysis/partitioning process

IV. Evaluation Results

Within this section, the SWS-ASE solution is evaluated with regard to selected quality criteria such as usability, reliability, efficiency, as well as to its applicability. All measurements were performed on a single Pentium 4 (3GHz, 512MB) PC with the Window XP (SP2) operating system. SWI Prolog 5.4.7 was used for all reasoning tasks.

A. Usability

In order to improve the usability aspects of the SE toolchain, SWS-ASE introduced several views, which provide a higher-level of abstraction for the underlying SWS-ASE processes and thereby hide unnecessary complexity. To give an example, Fig. 10 shows the graphical user interface for the specification of desired application features (feature modeling) in the context of workflow planning and execution.

As shown in Fig. 10, the given input (BookService, configuration) and desired output concepts (report) of a specific SE planning request can be added as boxes on the left. In this state, the composite SE process is not yet planned and available (cp. Fig. 7). On the right, the feature configurator is shown, which allows a tree-based and explicit selection of the desired application features.

Additional feature selection is facilitated using an iterative approach, whereby the list of possible features combinations available in the next step is updated and calculated dynamically. When the feature selection is completed, the "Generate workflow(s)" link launches the SWS-ASE process and automatically generates a valid workflow based on the given strategy parameters. Within the BookService scenario, the productivity of the SE process was significantly enhanced via feature modeling view and other views (such as depicting validation and registration). Compared to the traditional result times (using a manual engineering process), significantly improved engineering times and cycles were observable (deployment request creation, automated workflow planning phase, deployment execution). Other results showed that usability improvements occurred with regard to overall application management and maintainability. The transparent dependency/relation checks and automated composition process provided a more simplified engineering process, which reduced the cognitive demands on user/developer. Additionally, as shown in the BookService scenario and through usage within Siemens AG, SWS-ASE resulted in less error-prone and more reliable SE processes.

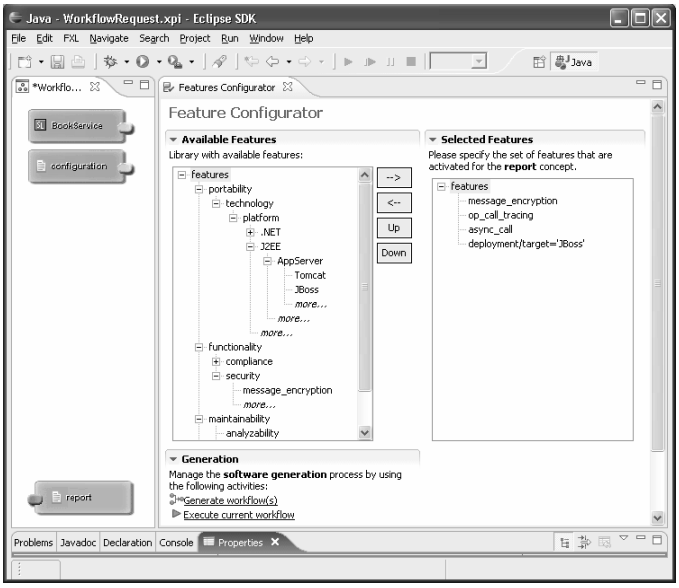

Figure 10. Feature configurator view

B. Reliability

To enhance reliability, the SWS-ASE approach reuses and combines functionality provided as SWS. As observed in the examined use case scenarios, the reliability of the generated workflows depends fundamentally on the accuracy of the semantic service descriptions. In order to improve reliability and accuracy, SWS descriptions are tested and partially validated with the analyzer tool of Fig. 9 at the point of registration. Accuracy tests are comprised that a start document D (which is assigned to the vertex c_0) is analyzed before the execution of the service and a result document D' (vertex c_1) is analyzed after the execution. In order to validate the semantic description, the set of detected features after service execution $F':=a(P, D')$ is compared with the result of the union $F_1 \cup F_2 = F$, whereby $F_1:=a(P, D)$ and $F_1:= f(W)$, $W=c_0c_1$. If they do not match, the applied rules R, which are used in the workflow W via the function f, are invalid. If so, the service description is invalid and was excluded from further workflows.

Additionally, SWS-ASE provides the capability of indicating the direction and degree of influence to quality characteristics of any given feature choice given the appropriate ontology and rules, thus allowing the impact of a choice to be seen as to how it affects the overall quality choices. For example, the choice of asynchronicity potentially enhances scalability and efficiency, while potentially reducing response time and maintainability. These quality tradeoffs could thus be weighed more explicitly by an engineer, and various quality priorities and optimizations could be selected to achieve a certain goal.

Other aspects that affected SE reliability included the SWS-ASE process and strategies, the domain ontology, SWS-ASE infrastructure (e.g., the service registry), and the reliability of any given tool service. Although not all of these aspects were addressed and covered by the solution, a general improvement in the reliability of the created application software was

observed. Compared with a manual SE process, fewer deployment and runtime errors occurred, due to the aforementioned dependency and relation checks within the creation phase of SE toolchain.

C. *Efficiency*

Response time and scalability measurements were applied to two typical search problems that occur in the planning phase:

- Find the first workflow that fulfills the request.
- Find the set of all possible workflows.

The underlying complexity and methodology of these request types was described in the preceding sections. As shown in Table II, the series of measurements for the BookService deployment request investigates the response times for the first reasoning result. In this test scenario, the service registry contains a set of 50 typical SE services. Additionally, the number of possible services within a workflow is restricted to a maximum of 10.

TABLE II. REASONING MEASUREMENTS (1)

Features							1st result time [s]	No. of valid workflow alternatives
op_call tracing	profiling	sync call	async call	message encrypt	deploy JBoss	deploy .NET		
-	-	-	-	-	-	-	0.26	530
-	-	-	-	-	x	-	0.30	162
x	-	x	-	x	x	-	0.92	22
-	x	-	x	x	-	x	0.89	24
-	-	-	-	-	x	x	n.a. [213.21]	0

Within the table, the Features columns represent possible random feature selections that are included in five variants of the BookShop deployment scenario, where 'x' means that a certain feature is requested. For example, the 'x' in row 2 and column 6 limits the deployment to a JBoss application server. For each variant, the "1st result time" shows the time in seconds until the first workflow for a specific scenario was created, whereby the "No. of valid workflow alternatives" gives the overall number of workflows that could theoretically fulfill the request. As expected, the table illustrates that as more features are requested, more time is needed to calculate a workflow. Additionally, as shown on the last row, the parallel deployment on two different platforms was defined as an illegal feature combination, which causes an initial result time of 213.21 sec. This occurs because all possible combinations of services have to be checked before the reasoner can decide that no valid workflow exists. In a practical scenario, a timeout (e.g., 5s) is introduced which makes use of the observation that an initial workflow variant is usually available quite early.

The second measurement series illustrates the scalability properties of the search algorithm with respect to the number of services based on the feature set in the 3rd variant (row) of Table II. In this scenario, the 1st column of Table III contains the number of services within the service registry which now varies from 15 to 50. The 2nd column states the response time until the first valid workflow was found. Additionally, the 3rd and 4th columns provide the time required until all possible workflow variants were available, whereby the 4th column does not restrict the number of services to 10 within a workflow in order to illustrate the effect of planning strategy constraints.

As depicted in Table III and from section III.B.4, the calculation time grows exponentially with the number of services that are included in the workflow-planning phase. In the BookService planning scenario, the complexity for the calculation of all workflows causes unacceptable results when more than 15 services are included. In contrast, an initial workflow variant is available after an acceptable period of time (e.g., 0.92sec for 50 services), which would be sufficient for most purposes.

TABLE III. REASONING MEASUREMENTS (2)

No. of Services in Registry	time [s] to calculate the 1st workflow result	time [s] to calculate all workflow variants	
		Max. 10 services in workflow	No restriction (unlimited services)
15	0.41	6.69	7.67
20	0.48	51.09	113.78
25	0.56	80.01	285.70
30	0.64	109.97	n.a.
40	0.78	159.54	n.a.
50	0.92	210.12	n.a.

Table IV addresses the performance aspects of the analysis process that is described in section III.B.4. In this context, several tests where applied using 3, 9 and 15 feature types in combination with document sizes of 250, 500, 1000 and 2500 lines of code (LOC). The measurements show that the time to analyze documents is independent of the number of found feature instances. The process of detecting feature patterns grows linearly with the number of available feature types and with the LOC of the input document. Thus, the underlying process of feature recognition scales well and is not a critical part within the overall SWS-ASE process.

TABLE IV. ANALYSIS PROCESS MEASUREMENTS

No. of feature types	250 LOC		500 LOC		1000 LOC		2500 LOC	
	Time [ms]	Feature instances found	Time [ms]	Feature instances found	Time [ms]	Feature instances found	Time [ms]	Feature instances found
3	102	0	172	0	329	0	682	0
9	105	12	179	25	343	12	706	12
15	110	36	189	75	350	36	727	36

D. *Applicability*

To achieve the Semantic Web SE domain grand vision of the global integration of the tool infrastructure, new areas of difficulty must still be addressed by the Semantic Web and SE community. Fundamentally, all of the open issues in the intersection of the Semantic Web and Automated SE affect the general applicability of the presented SWS-ASE approach. Such issues include, e.g., versioning management of software artifacts/assets and their related ontologies and interdependency rules, e.g. with respect to compatibility. For workflow planning, the NP-hard issue remains, although improved heuristics and algorithms could help. Moreover, SWS workflows are currently tied to WSDL documents that could

change over time and cause the generated workflows to become stale. In the context of SWS-based distributed workflows, security and trust play a significant role. Other current issues include QoS, SLAs, registering/finding services globally, usage license agreements, and business models for service providers. Additionally, standardization work is needed for semantic descriptions (e.g., WSDL-S) and upper/lower ontologies within the SE process. Thus, the general applicability of the SWS-ASE approach depends on the widespread adoption and usage of the SWS paradigm to aggregate tool functionality and achieve greater automation. Nevertheless, the SWS-ASE results show that the use such an approach can yield promising quality improvements for common SE tasks (such as automated toolchain composition and execution), that major parts of the SWS-ASE underlying vision can already be applied.

V. Related Work

SWS-ASE combines technologies from various research areas and thus facilitates interdisciplinary collaboration. Within this section, several approaches and technologies are presented that cover certain parts of the SE process and/or that could help within the context of the presented solution.

ASE and integrated toolchains are currently being approached on various levels. IDEs such as Microsoft Visual Studio and the IBM Websphere Studio support the vision behind software factories [14]. Integrating various development tools into one homogeneous adaptable and configurable CASE tool environment requires the usage of standardized interchange and description formats and a well-defined meaning on which each component can rely. Although Eclipse relies on OSGi, which furthers a standards-based, customizable, and potentially distributed development environment, the usage of SWS-ASE and the underlying semantic web technologies in this context could enhance collaboration and interoperability between SE tools. Distributed tool infrastructure is gaining in popularity, for instance Apache Maven. While Maven provides consistent conventions and dependency management across Java projects, its underlying metadata and process is not standardized and interoperable across platforms. Additional research work in the area of integrated, distributed toolchains include meta-model [15], peer-to-peer [16], and agent [17] approaches. While addressing integration between tools at different levels, SWS-ASE has a strong focus on semantic and automatic composition.

OMG's MDA and other MDSD approaches provide support for automation and generation from the model through the execution cycle. Currently, MDA does not address the feature dependency implications that correlate with toolchain interactions. With regard to this issue, SWS-ASE can support MDA's vision via structural and behavioral dependency and relation analysis of the toolchain context with a standardized, distributed, and interoperable SWS approach.

In the context of this paper, feature modeling supports the conceptual abstraction and description of relationships between distinguishing characteristics of SE artifacts or assets. Implementations such as XFeature and pure:variants support variant management and product families, but because they do not use semantic-based interoperable standards such as OWL and SWS, they are limited in the degree of automation and distributed inter-platform operability.

Related Semantic Web research involves composition and ontologies. For example, AI Planning research for automated composition includes rule-based or Hierarchical Task Network (HTN) approaches. Although pure HTN planners such as SHOP2 perform well in certain scenarios, scalability can become an issue for complex knowledge bases. An HTN-related approach that uses additional heuristics [12] to address complexity is OWLS-Xplan [9], which shows that better solutions are possible for certain problem spaces. [18] gives an overview of the current state-of-the-art and evolving AI Planning methods, which could enlarge the spectrum of planning strategies available to the SWS-ASE approach to handle the complexity inherent in the SE domain. Another Semantic Web and SE issue is the lack of available SE domain ontologies, which is currently being pursued [19].

For workflow description, possibilities include BPEL4WS and Microsoft Windows Workflow Foundation (WWF). Ideally, workflows across SE toolchains should be able to handle dynamic aspects (consider AO4BPEL [20]) and be capable of self-adaptation, for which synergies with stochastic optimization [21], AI planning and autonomic computing research are conceivable.

VI. Conclusion & Future Work

Due to the multiple challenges confronting SE, there is a demand for greater automation and better toolchain integration. Currently, the missing semantic annotation as well as limited use of the interoperability paradigms makes it difficult to aggregate the underlying parts and thereby achieve a higher-level automated SE process. Despite advances in this area, multiple and often hidden dependencies, implicit decisions, and manual steps comprise a typical part of the current engineering cycle. The ensuing mistakes, needless complexity, inefficiencies, and unmanageability affect the quality of the software product and the development process.

In this paper, we presented the SWS-ASE approach. By preparing and registering tool functionality as SWS, an automated SE process was introduced that is able to provide a way of amalgamating heterogeneous, distributed tool functionality in a standardized way. The basic applicability of this approach was shown via the BookService example scenario. In this context, user-specific views such as the Feature Configurator were provided to improve usability by hiding technology-specific details and enabling feature-driven application development. The results showed that typical feature selections could be reasonably handled when the appropriate algorithms and heuristics are applied together with constraints that reduce the set of possible combinations. Fundamentally, the enhanced automation of the SE process yielded improved usability, reliability, and efficiency.

Some inherent risks of the SWS-ASE are that the quality of the overall process strongly depends on the correctness of the syntactic and semantic descriptions, and that the number of relationships (regarding possible I/O connections, feature

cross-dependencies, etc.) increases exponentially with the number of involved services. To mitigate these risks, it is necessary, e.g., that the SE community provide a standardized process for the creation and maintenance of SE domain models (e.g., to handle different versions). Additional research (i.e., improved strategies and algorithms) of the SWS community is needed in the field of workflow planning and complexity handling, where the NP-hard criterion remains an issue. Nevertheless, the SE domain has much to gain from the SWS vision and the enhanced automation capabilities it could bring. In this context, by improving the integration of the software tool infrastructure with semantically-enhanced tools, and by automating the planned workflows based on desired features, the SWS-ASE approach shows that significant benefits are realizable that support the overall quality of the software product and process.

Future work in the context of SWS-ASE will include the improvement of the tool automation spectrum for semantic analysis, pattern matching, partitioning, and description generation. To enable more dynamic and adaptable workflows, new strategies for dynamic semantic queries will be explored as well as the application of transactional approaches.

ACKNOWLEDGMENT

We would like to thank Klaus Jank (Siemens AG), Donald Kossmann (ETH Zurich) and Alexander Schill (TUD) for their continuous support of our research work.

REFERENCES

[1] Berners-Lee, T., Fischetti, M.: Weaving the Web – The original design and ultimate destiny of the World Wide Web, by its inventor, Harper San Francisco; 1st edition, September 1999.

[2] Akkiraju, R., Farrell, J., Miller, J.A., Nagarajan, M., Schmidt M-T., Sheth, A., Verma, K. Web Service Semantics - WSDL-S, Technical Note, Version 1.0, April 2005, http://www.alphaworks.ibm.com/g/g.nsf/img/semanticsdocs/$file/wssemantic_annotation.pdf

[3] Tetlow, P., et al.: Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering, W3C Working Draft, Oktober 2005, http://www.w3.org/2001/sw/BestPractices/SE/ODA/ .

[4] International Standard ISO/IEC 9126. Information technology -- Software product evaluation -- Quality characteristics and guidelines for their use, International Organization for Standardization, International Electrotechnical Commission, 1991, Geneva.

[5] Smith, W.D., TPC-W: Benchmarking An Ecommerce Solution, Revision 1.2, http://www.tpc.org/tpcw/TPC-W_wh.pdf.

[6] Software Engineering Body of Knowledge (SWEBOK), http://www.swebok.org, Access date: November 1st, 2005.

[7] Kossmann, D., Reichel, C.: SLL: Running my Web Services on Your WS Platforms, Proceedings of ICWS 2005, Industry Track, July 2005. Also see www.open-xl.org.

[8] Medjahed, B., et al.: Composing Web services on the Semantic Web. The VLDB Journal, Volume 12 , Issue 4, November 2003, pp. 333 – 351.

[9] Klusch, M.; et al.: Semantic Web Service Composition, Planning with OWLS-Xplan. 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web, Arlington VA, USA, 2005.

[10] Sirin, E., et al.: HTN Planning for Web Service Composition Using SHOP2, Journal of Web Semantics, Volume 1, Issue 4, 2004.

[11] T. Bylander. The computational complexity of propositional STRIPS planning. Artificial Intelligence, 69(1-2):165--204, 1994.

[12] J. Hoffmann, B. Nebel: The FF Planning System: Fast Plan Generation Through Heuristic Search, Journal of Artificial Intelligence Research, Volume 14, 2001, pp. 253 - 302.

[13] Florescu, D., et al.: XL: An XML Programming Language for Web Service Specification and Composition, Computer Networks Journal, Volume 42, Issue 5, August 2003.

[14] Greenfield, J.: Software Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools, Wiley, 1st edition, August 2004.

[15] Burmester, S., et al.: Tool integration at the meta-model level: the Fujaba approach, STTT, Volume 6, Number 3, August 2004, pp. 203-218.

[16] Hansen, K.: Activity-centred tool integration. Using Type-Based Publish/Subscribe for Peer-to-Peer Tool Integration, Proceedings of the ESEC Tool Integration Workshop, 2003.

[17] Corradini, F., et al.: An agent-based approach to tool integration, STTT journal, Volume 6, Number 3, August 2004, pp. 231-244.

[18] Rao, J., Su, X.: A Survey of Automated Web Service Composition Methods, In Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004, San Diego, California, USA, July 6th, 2004.

[19] Wongthongtham, P.: Software Engineering Ontologies and Their Implementation, Proceedings of the IASTED Conference on Software Engineering, 2005, pp. 208-213.

[20] Charfi, A., Mezini, M.: Aspect-Oriented Web Service Composition with AO4BPEL, Proceedings of ECOWS 2004, LNCS 3250. September 2004.

[21] Doshi, Prashant, et al: Dynamic Workflow Composition using Markov Decision Processes, ICWS'04, p. 576