A Knowledge-Based System Approach for Extracting Abstractions from Service Oriented Architecture Artifacts

George Goehring, Thomas Reichherzer, Eman El-Sheikh, Dallas Snider, Norman Wilde, Sikha Bagui,
John Coffey, Laura J. White
Department of Computer Science
University of West Florida
Pensacola, Florida, U. S. A.

***Abstract*—Rule-based methods have traditionally been applied to develop knowledge-based systems that replicate expert performance on a deep but narrow problem domain. Knowledge engineers capture expert knowledge and encode it as a set of rules for automating the expert's reasoning process to solve problems in a variety of domains. We describe the development of a knowledge-based system approach to enhance program comprehension of Service Oriented Architecture (SOA) software. Our approach uses rule-based methods to automate the analysis of the set of artifacts involved in building and deploying a SOA composite application. The rules codify expert knowledge to abstract information from these artifacts to facilitate program comprehension and thus assist Software Engineers as they perform system maintenance activities. A main advantage of the knowledge-based approach is its adaptability to the heterogeneous and dynamically evolving nature of SOA environments.**

Keywords—expertise; rule-based system; knowledge-based system; service oriented architecture; SOA; software maintenance; search tool.

I. SOA, Maintenance And The Role Of Expertise

Rule-based methods have been very effective in supporting decision making in many complex domains. Can they also assist Software Engineers in dealing with the emerging complexities of Service Oriented Architecture (SOA) applications?

SOA is not a single software architecture, but rather a style for constructing complex systems, especially those that need to cross organizational boundaries. SOA systems, often called *composite applications*, typically resemble Fig. 1.

An organization, whether governmental, non-profit, or private, finds that it needs to work with other organizations to carry out key workflows.

For example fulfilling a purchase order requires getting stock from a partner company, planning employee travel involves reservations on several airlines, or providing a doctor with a patient's medical history entails assembling information from many medical records systems.

As shown in Fig. 1, in a SOA architecture the software to support these workflows is organized as *services* having defined interfaces, running on different nodes and communicating via message passing. Some of these services will be owned and managed by the home organization but others will belong to partners or be offered by commercial vendors.

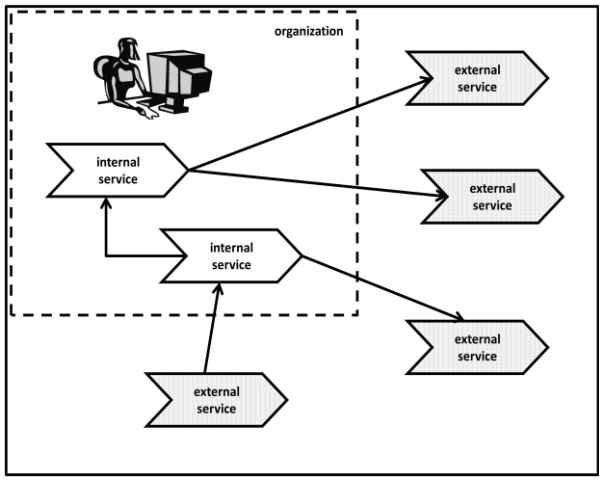

Fig. 1. A SOA Composite Application

Most commonly the Web Services group of standards is used to define the service interfaces and protocols [1]. In theory, these standards are supported by a broad group of providers so that services can interoperate across many different programming languages, operating systems, and data definition schemas. However, the standards have turned out to be both very complex and very loose, leading different implementers to create services and interfaces having vastly different styles.

SOA composite applications began to appear at the start of the twenty first century and by now are very widespread. They have faced many technical and managerial difficulties, but perhaps none will be more difficult than the challenge of *software maintenance* as these systems begin to age. Traditionally, maintenance of large software applications has been particularly expensive and slow because typically:

a) There is a large code base of existing, legacy software.

b) To make changes safely, scarce and expensive Software Engineering personnel must first invest time to understand that existing software.

c) Turnover of such personnel leads to loss of human knowledge and the application gradually slides into a state sometimes called "servicing" in which only very limited changes may be safely attempted [2].

The essential reason for the cost and delays of software maintenance is thus the difficulty of acquiring and sustaining necessary Software Engineering expertise. As several authors have pointed out, sustaining that expertise for SOA may be even harder than with earlier application styles [3] [4] [5] [6] [7] [8]. The challenges include:

1) The heterogeneity of SOA applications, so that maintainers may need expertise in many different languages, environments, and implementation styles.

2) The distributed ownership of services, so that for business reasons source code or key documents may not be made available to the maintainers.

3) Poorly coordinated changes, as the different service owners are driven by different business needs, leading to crises and to multiple fielded versions of each service.

SOA Software Engineers will thus have to respond to continual and often unpredictable change as they maintain large heterogeneous applications exhibiting a bewildering variety of programming styles. This research explores how knowledge-based methods can help provide the necessary expertise to help SOA systems evolve at reasonable cost.

In this paper we describe a knowledge-based approach to this problem, in which a rule-based system is used to enhance search techniques so that a Software Engineer can more rapidly understand a given composite application. The rule-based system generates *abstractions*, snippets of information that summarize complex application relationships to provide context quickly. The main benefit of the rule-based method is adaptability; different application styles and changing environments may be handled by relatively simple modifications to the rules. Thus a rule set can itself dynamically evolve as the composite application evolves to meet changing needs.

In the next section the article reviews related work followed by a presentation of an illustrative example to motivate the need for SOA abstractions. Then it describes the design principles appropriate for search in a SOA context, discusses the knowledge-based approach to SOA abstraction, and presents the results of an evaluation case study. The article concludes with a summary of key contributions and suggestions for future work.

II. Related Work

Although little literature is available regarding the use of rule-based systems for SOA system maintenance, rule-based systems have been applied more broadly to software understanding. Canfora and Di Penta [4] describe two tools, Design Maintenance System [9] and TXL [10] which parse source code and, through rule-based transformations, produce artifacts that facilitate program understanding. Braun [11] describes a server-based analysis system based upon rules that is designed to play a role in configuration management of software. The idea is that checked-out versions can be subjected to rule-based checks for various attributes before they are committed to a version control system.

Rule-based information extraction akin to the idea of summarizing software abstractions in the current work appears to be an area of increasing interest. Zaghouani [12] describes a system for named entity extraction from text in natural language processing. Wang [13] describes named entity extraction with rules and a machine learning approach using "conditional random fields." Michelakis et al. [14] describes rule-based information extraction in which structured objects are extracted from text, based on user-defined rules.

Research on tools to support maintenance of SOA systems has been fairly limited. Most of the proposals involve dynamic analysis, usually of a trace from a running system. A group from IBM has described a tool called Web Services Navigator that uses dynamic analysis to provide five different views of an executing system [15]. Two papers describe ways of locating user features within a SOA system. One approach produces a sequence diagram showing the feature [16] while the other does an analysis of dynamic call trees [17]. Halle et al. have a somewhat different approach that starts from a hypothesized service contract and automatically sends a series of trial invocations to see if the service actually conforms to the hypothesis [18]. Dynamic analysis is a powerful approach to understanding a system; the main difficulty is that it is frequently impractical to gather the needed data from a large system running across multiple nodes.

III. SOA Maintenance Challenges: An Illustrative Example

To illustrate the problem of understanding SOA, consider an example from WebAutoParts.com, one of the composite applications in our Open SOALab collection of resources for SOA teaching and research [19]. WebAutoParts.com (Fig. 2) is a hypothetical online automobile parts supplier that uses external services to facilitate agile development. As is true for many SOA composite applications that are based on the Web Services standards, the main artifacts that describe WebAutoParts are BPEL program code, WSDL service interface descriptions and XSD data type definitions.

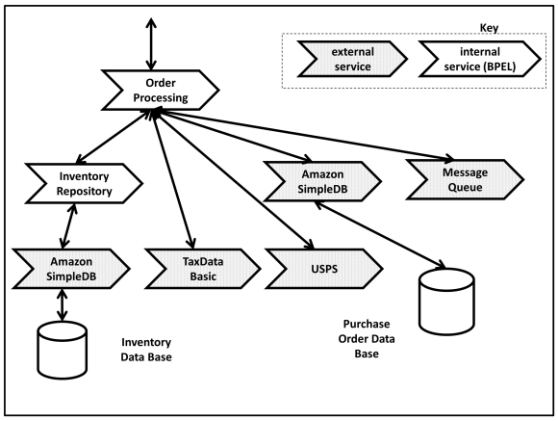

Fig. 2. Webautoparts.Com - Order Processing Workflow

BPEL, the Business Process Execution Language, is an XML formatted language that describes how services are orchestrated together to perform a complete workflow [20]. Each BPEL program itself becomes a service when it is interpreted on an application server. WSDL stands for Web Services Description Language [21]. WSDL files, which again have an XML format, describe the interface that a service presents to its clients. XML Schema Descriptions (XSDs) are an XML language used to describe the data types for the message data that is passed between services [22]. The data type descriptions for a particular service may either be incorporated into the <types> section of the service's WSDL file or else included from an external XSD file.

WebAutoParts has an order processing workflow shown in Fig. 2. There are two "stubbed" in-house services written in BPEL (Order Processing and Inventory Repository) and four external services from three well-known vendors:

- Amazon Web Services - *Amazon Simple DB* (database) and *Message Queue* (message queuing)
- StrikeIron.com - *Tax Data Basic* (sales tax rates)
- Ecocoma - *USPS* (shipping costs)

In this workflow, an incoming order is first checked against inventory to confirm that it can be processed. Then sales tax is computed based on the rules of the state where the customer resides. Shipping costs are then computed and added and finally the order is added to a message queue to be picked up by the order fulfillment service. While the WebAutoParts application does not actually execute, it consists of syntactically correct BPEL code which deploys successfully to the Ode BPEL environment along with XSD and WSDL documents typical of current industrial practice.

Suppose a Software Engineer unfamiliar with this application is trying to implement a change to the database design and needs to know what data is passed when Order Processing checks inventory levels. If he has extensive BPEL/Web Services experience he might figure this out using a series of searches (Fig. 3). In these searches he must match the names appearing in different XML elements and navigate up and down the containment hierarchy of these elements:

1) Search the Order Processing BPEL file to find the <invoke> tag that is checking inventory. That provides him a partnerLink. Then search the partnerLinks to get the partnerLinkType which turns out to be IRepositoryLinkType.

2) However, there is no indication of which service implements this link type, so the Software Engineer now searches all the WSDL documents for that link type. He will find it in InventoryRepository Artifacts.wsdl with a pointer to the WSDL portType for the service. The portType in turn gives the operation and its input and output message names. A further search on the message name reveals that the message contains an element called inventoryQuery.

3) *However inventoryQuery is not defined within the WSDL so the Software Engineer now has to search XSDs to eventually locate the definition of inventoryQuery, determine its type, and from its type finally conclude what data fields are being passed.*

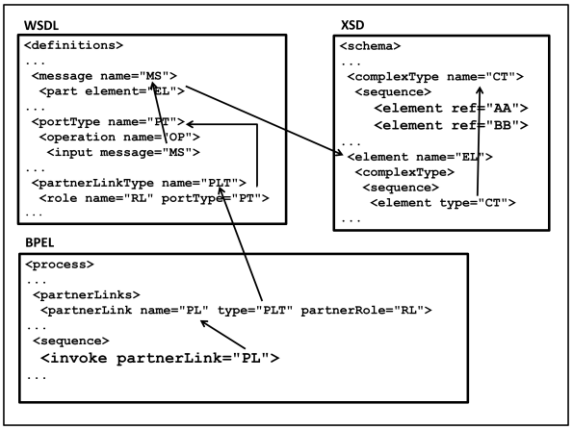

Fig. 3. Searching BPEL, Wsdls And Xsds

Even for a Software Engineer who is an expert in Web Services, tracing such chains of relationships requires a tedious and error-prone sequence of searches. Furthermore, the heterogeneity of SOA services will mean that expertise may not generalize well from one composite application to another. Each such application may use a different combination of technologies and apply them in different ways. There are, for example, many textually different ways to describe essentially the same message data using WSDLs and XSDs. Worse, the Web Services standards themselves are evolving so it is likely that a maintainer will encounter fielded systems based on different versions. Finally, since the WSDLs, XSDs, and configuration files that describe a composite application are often machine-generated, they contain "clichés" or patterns that are peculiar to a particular development environment. For example, an XSD generated by Microsoft's WCF framework contains five-tag sequences of XML to simply declare a void return type for an operation [23].

There is a lot of information contained in the artifacts describing a SOA composite application. Experts with long application-specific experience may be able to navigate these artifacts, but such experts will be scarce. Thus, the focus of this research is to develop a rule-based system that mimics expert reasoning on the SOA artifacts to provide useful information for a wider range of Software Engineers lacking specific knowledge in handling the artifacts.

IV. Intelligent Search for SOA Maintenance

Intelligent search tools can help users find the kinds of information in SOA composites that maintainers may need. Search tools based on text matching are usable on a variety of document types making them a good fit for the heterogeneous world of SOA composite applications. Our group has been conducting research on the application of intelligent search for SOA maintenance using `SOAMiner`, a prototype SOA-specific search engine. Case studies with different groups of academic and real-world programmers have been exploring "what SOA maintainers will want to know" [23] [24].

The results of these studies have shown that participants found it easy and natural to search a large corpus of artifacts from a SOA composite application. They quickly found relevant snippets of information, such as all the XML tags containing a keyword such as "inventory". However search

identified each snippet in isolation and did not show its context within the application as a whole. In some cases it was sufficient to simply show more of the surrounding text, but it is clear that for other problems a Software Engineer would need to make a tedious sequence of searches such as those in the example given earlier.

We conclude that, for SOA, search needs to be enhanced with a process of abstraction. For example, a search should take the user to relevant fragments of a BPEL, WSDL or XSD, and then provide a higher-level abstraction that shows how that fragment fits into a wider reality. A difficulty, of course, is that in SOA's open environment the relevant abstractions will vary from system to system and over time as standards, practices, and tools change.

Thus we need an adaptive and dynamic abstraction mechanism to complement SOA search. An ideal tool would index the collection of artifacts from a composite application and:

1) *Provide abstraction-enhanced search where it can.*
2) *Provide useful text-based search where it cannot.*
3) *Allow the definition of additional abstractions so that more and more searches can be moved into the first category.*

Such a tool should be flexible to adapt to a wide range of SOA artifacts from different environments and allow for the inclusion of new abstractions as they are discovered.

V. A KNOWLEDGE-BASED SYSTEM FOR SOA ABSTRACTION

Knowledge plays a key role in achieving intelligent behavior. Knowledge-based systems capture human knowledge, represent it in a machine readable form, and facilitate reasoning with it for solving problems. The following describes our approach to capture human expertise in SOA code analysis and to use that expertise for analyzing SOA artifacts and providing intelligent search support.

A. Rationale for Using a Rule-Based System

Rule-based systems have traditionally been used to capture human expertise as a set of rules to draw conclusions from chains of rules applied to initial facts stored in a working memory. As the rules execute, new facts are being generated and added to the working memory causing other rules to execute. Eventually, the rules have completely transformed the facts in memory and no rule can execute. The working memory contains the conclusions that the rules derived. This flexible control, inherent to rule-based systems, differs from predefined control structures found in programs of traditional programming languages. Rules can be easily modified or extended to adjust the performance of the rule-based system. Thus, rule-based systems are an ideal method for dealing with the heterogeneous nature of SOA applications and their evolving artifacts, to identify and extract abstractions automatically and make them available for inspection.

Through experiments and case studies involving domain experts we create a set of rules that identify abstractions within the SOA artifacts, and extract and transform these abstractions into machine-readable representations. In essence, the rules capture an expert's knowledge and skills to identify useful excerpts of information relevant to software maintenance tasks and the reasoning engine automates the process of the expert's analysis of SOA artifacts by executing chains of rules on the artifacts once they are committed to the engine's working memory.

B. System Architecture

Fig. 4 shows the system architecture of the knowledge-enhanced search tool. The tool is composed of an *XML annotator*, a *search indexer*, and a *reasoning engine*. It processes XML Files since many SOA artifacts have XML structure (WSDL, XSD, BPEL and many configuration files). As a first step, the tool annotates every element in the input XML files with a unique identifier so that it can be referenced in the reasoning engine and during searches. After annotation, the files are loaded both into the search indexer and the reasoning engine. This engine runs the DROOLS Expert rule-based system to identify and construct abstractions from the input sources [25].

The engine executes rules on XML elements in the imported files to identify abstractions existing within the artifacts and build them in working memory. As abstractions are committed to the working memory as temporary results the rules may subsequently discover new abstractions and relationships between them. Finally after all rules have fired, working memory is queried to store the abstractions in files that can then be displayed in response to searches in support of maintenance tasks. Each abstraction is formatted as an XML snippet that includes constituents and relations from the SOA artifacts to model the abstraction. The final output is in the form of three XML files, one containing the set of abstractions, another containing cross-references when one abstraction refers to another, and a third describing the search index for the Apache Solr search platform [26].

C. Design of the Knowledge Base

Our case study produced three types of abstractions to support maintenance activities: A) data type summaries, B) services, and C) BPEL invoke relationships. Based on these findings, we analyzed artifacts from the WebAutoParts SOA composite application to look for abstractions and to identify the information that is needed to produce them. From this information, rules and representations were built that match XML elements in the SOA artifacts and transform them into new representations to describe the different abstractions.

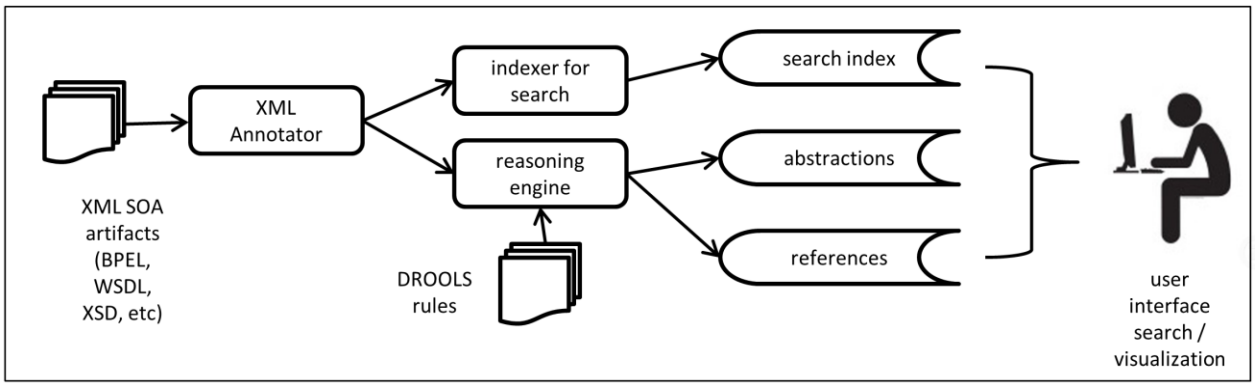

Fig. 4. Architecture For Knowledge-Based Search Tool

In order to make the program extendable, XML elements from the SOA artifacts are loaded into a generic structure called an Entity object that holds each element's type, as well as all of its attributes This structure is then used by the DROOLS rules, which contain the knowledge of how to operate on specific vocabularies of XML, to make transformations leading to the construction of Abstraction objects added by the rules to the working memory. Abstractions are subclasses of Entity to ensure that each Abstraction is also an Entity. Finally, Dependency objects store relationships between two Abstraction objects as established by the DROOLS rules. For example, a Dependency object may describe a relationship that exists between a message in a service abstraction and a data type summery abstraction. Each Entity has a Location, which corresponds to a single input file. Location objects also store statistics about the number of Abstractions identified in imported SOA artifact files. The entire object model is depicted in Fig. 5.

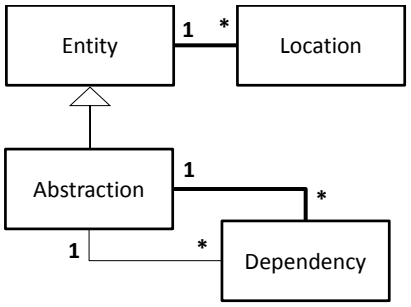

Fig. 5. Object Model For Storing Xml Elements

The rules perform multiple transformations on the XML elements, extracting patterns and tracing the links between complex structures it identifies in the artifacts. The conditional parts of each rule matches against the objects in working memory and its specific values. The action part generates new objects. The rule engine executes the rules until no further transformations can be performed and all abstractions have been identified. Since a generic structure was used for representing XML nodes, additional DROOLS rules may be easily added to the system for new XML vocabularies.

The initial rule set included six rules, three that work together for creating data type summaries, two that create service abstractions, and a single rule that generates high-level BPEL invoke relationships. The three data type summaries rules include a general preprocessing rule, a rule for generating Complex Type Sequences (CTS) and a rule for generating Complex Element Sequences (CES) (details to follow later). The two rules for generating service abstractions perform two independent steps. The first rule looks for services and its operation and the second rule looks for messages associated with operations.

D. Example Application

To show the expressive power of the rule-based approach, consider the problem of identifying which services a BPEL program actually calls. This is not explicit in the code since, to allow for loose coupling of services, BPEL only contains "partner links" which may be resolved to a specific service on deployment or even at runtime.

Table I shows the DROOLS rule and sample fragments of the BPEL and WSDL elements that it operates on. The first part of the table shows the DROOLS rule (lines 1 – 15) and the second part shows the XML fragments from the BPEL and WSDL files (lines 16 – 28). Specifically:

- On line 3, the rule accesses a BPEL partnerLink such as the one on line 16.
- Lines 4 and 5 of the rule match the WSDL's partnerLinkType and role elements from lines 18 and 19 using the "IRepositoryLinkType" and "repository" values.
- On line 6 the rule locates the WSDL binding element of line 22 by matching on the "InventoryRepository PortType".
- Lines 7 and 8 of the rule match the WSDL's service and port elements (lines 23 to 28) using "InventoryRepositoryBinding".
- Finally on lines 10 through 14 the rule creates and stores a new abstraction with the name of the service, thus identifying the actual service called.

As can be seen, a Software Engineer could find it very tedious to follow this chain of relationships by hand, but the rule can abstract the chain to a simple conclusion: OrderProcessing calls InventoryRepository.

TABLE I. CREATION OF AN ABSTRACTION FROM RULES

DROOLS Rule

```
1  rule "High Level BPEL Partner Link Invokes Abstraction"
2  when
3     $plnk : Abstraction(type == "partnerLink")
4     $plnkType : Entity (type == "partnerLinkType" && getAttribute("name") == $plnk.getAttribute("partnerLinkType"))
5     $role : Entity (type == "role" && parent == $plnkType && (getAttribute("name").equals($plnk.getAttribute("partnerRole"))))
6     $binding : Entity (type == "binding" && getAttribute("type") == $role.getAttribute("portType"))
7     $port : Entity (type == "port" && getAttribute("binding") == $binding.getAttribute("name"))
8     $service : Entity (type == "service" && hasChild($port))
9  then
10    Abstraction root = new Abstraction($plnkType);
11    root.setType("partnerLinkType");
12    root.setAbbreviation("PLType");
13    root.addAttribute("name", $service.getAttribute("name"));
14    $plnk.addChild(root);
15 end
```

Excerpt from OrderProcessing.BPEL

```
16  <bpel:partnerLink name="inventoryRepositoryLink"   partnerLinkType="ns2:IRepositoryLinkType" partnerRole="repository" />
```

Excerpts from InventoryRepositoryArtifacts.WSDL

```
17 <!-- PARTNER LINK DEFINITION -->
18  <plnk:partnerLinkType name="IRepositoryLinkType">
19    <plnk:role name="repository" portType="tns:InventoryRepositoryPortType"/>
20 </plnk:partnerLinkType>
       •••
21 <!-- BINDING DEFINITION -->
22   <binding name="InventoryRepositoryBinding" type="tns:InventoryRepositoryPortType">
       •••
23 <!-- SERVICE DEFINITION -->
24 <service name="InventoryRepository">
25    <port name="InventoryRepositoryPort" binding="tns:InventoryRepositoryBinding">
26       <soap:address location="http://WebAutoParts.com:9990/InventoryRepository" />
27    </port>
28 </service>
```

VI. EVALUATION CASE STUDY AND RESULTS

To illustrate the power and flexibility of the knowledge-based approach to SOA abstraction, we performed an evaluation case study using two different SOA composite applications.

The first case study involved the WebAutoParts example mentioned earlier, and the second involved a Travel Reservation Service originally included as a tutorial example with the NetBeans IDE, version 6.0. Both applications consisted of BPEL orchestration code which invokes services defined by WSDLs and XSDs. Table II shows the dimensions of each application.

In our case studies for SOA search ([23], [24]), Software Engineers had identified several different kinds of abstractions that they thought would be useful. For the evaluation case study of the knowledge-based system, we used the three most prominent of these:

A. Tree representation of a service

The description of a service in a WSDL is dispersed and usually needs to be read "bottom up" starting from the port element at the end of the file and proceeding upward through binding, portType, and message elements to arrive at the input and output message structures [1].

TABLE II. SOA APPLICATION COMPOSITION

File Type	WebAutoParts		Travel Reservations	
	Files	Lines	Files	Lines
BPEL	2	189	1	417
WSDL	6	2433	4	524
XSD	2	64	1	17034
Total	10	2686	6	17975

Software Engineers requested a more compact, top-down view of a service, its operations, and its input and output messages.

Fig. 6 gives an example for the USPS shipping-cost service abstraction from the WebAutoParts application.

B. Compact data type summaries

Data handled by a service can be described in many different locations: directly in message structure, in the "types" section of the WSDL, or in imported XSD statements. In turn, each element or type can reference other elements and types, so the Software Engineer trying to understand data must often pull together information from many different parts of several different files. Not surprisingly, participants in our studies requested a more compact summary so that the complete structure could be viewed in one place.

```
SERV - USPS_Service
  OP - GetUSPSRate
    OUT-MSG - GetUSPSRateSoapOut
      ref - GetUSPSRateResponse
    IN-MSG - GetUSPSRateSoapIn
      ref - GetUSPSRate
  OP - GetExtendedUSPSRate
    OUT-MSG - GetExtendedUSPSRateSoapOut
      ref - GetExtendedUSPSRateResponse
    IN-MSG - GetExtendedUSPSRateSoapIn
      ref - GetExtendedUSPSRate
```

Fig. 6. Tree Representation Of The Shipping Cost Service

The two most common patterns for describing structured data in XSD are either as a <complexType> that can be reused in several places or directly in an <element>. Accordingly two kinds of data type summary abstractions were defined in the rule set: Complex Type Sequences (CTS) and Complex Element Sequences (CES). Fig. 7 gives an example of the InventoryQuery CTS used in WebAutoParts. The description of this element in the original XSD takes 12 lines distributed in different parts of the file. The CTS reduces that to the 5 contiguous lines of Fig. 7.

```
CTS - InventoryQueryItemType
  E - element - PartNumber
  E - element - Description
  E - element - UnitPrice
  E - element - NumberInStock
```

Fig. 7. Compact Abstraction Of A Complex Type

C. High-level BPEL invoke relationships

The example in Section II showed some of the complexities of tracing BPEL code. For our rule set we defined an "invoke operation" abstraction that traces from the <invoke> tag in the original BPEL to locate the actual service and operation being called. These "invoke operation" abstractions can be combined to give an approximation of the service call tree of the composite application. Fig. 8 shows an example recovered from WebAutoParts. Note the similarity to the workflow diagram of Fig. 2. For some services, such as USPS_Service, two links are shown because the service offers two different bindings for clients using different versions of SOAP or different transports. Statically, the BPEL cannot reflect which is in use.

```
OrderProcessing invokes:
  USPS_Service.USPS_ServiceSoap12.GetUSPSRate
  USPS_Service.USPS_ServiceSoap.GetUSPSRate
  TaxDataBasic.TaxDataBasicSoap.GetTaxRateUS
  MessageQueue.MessageQueueHttpsPort.SendMessage
  MessageQueue.MessageQueuePort.SendMessage
  AmazonSimpleDB.AmazonSDBPortType.PutAttributes
  InventoryRepository.InventoryRepositoryPort.checkInventory
InventoryRepository invokes:
  AmazonSimpleDB.AmazonSDBPortType.GetAttributes
```

Fig. 8. Services And Operations Called In Webautoparts

D. The evaluation study and its results

The starting point for the evaluation case study was an initial set of rules that had emerged while the knowledge-based system was under development. To guide that development we used our background expertise about Web Services in general, with WebAutoParts being a prominent running example. We wanted to see how hard it would be to adapt this set of rules when we moved to a second, less-familiar system. An independent evaluator who had not participated previously in the project inspected both WebAutoParts and TravelReservations composite applications by manually examining the corresponding BPEL, WSDL, and XSD files. The evaluator identified the services, data types, and invoke relationships which should have been discovered from his perspective. Anything perceived to be unusual or incomplete as assessed by the evaluator was marked as an "anomaly". The results are given in Table III.

Not surprisingly, since WebAutoParts was one of the examples used in developing the initial rule set, only 9 anomalies were encountered, and these fell into 3 categories. One CTS encountered by the evaluator was actually an extension of another data type; the <extension> element in XML schema may be used to add additional data items to a data structure, providing a form of inheritance. The initial rules were not sophisticated enough to identify this case, which only appeared once across both examples.

In another case the evaluator was surprised to see one CES that seemed to appear twice. In fact, two different services happened to use elements having exactly the same name. Perhaps the most interesting case was 6 CESs from one WSDL file which were correctly found, but without their structure. It turned out that this WSDL attached <documentation> tags to the input message of each service operation. These tags confused the rule that assembled the structure of the CES. This particular anomaly illustrates the heterogeneity of SOA implementation styles, with each service developer making different choices about where to place documentation.

More interesting was the Travel Reservations application where we saw even more the effects of heterogeneous implementation styles. The initial rule set correctly identified the large number of data types (CTS and CES) but encountered some significant variations in service and "invoke operation" abstractions.

TABLE III. EVALUATION RESULTS FOR THE INITIAL RULE SET

	Services	Operations	Messages	CTS	CES	Invoke
WebAutoParts						
- correct	7	44	88	74	135	6
- anomalies				1	8	
Travel Reservations						
- correct	4	12	16	543	172	0
- anomalies	3					6

Travel Reservations includes 4 distinct services, a "top level" BPEL orchestration service and 3 partner services representing airline, hotel, and rental car companies. In this application the services use an asynchronous "request/callback" message exchange pattern, unlike the synchronous "request/response" of WebAutoParts. This means that the top level service provides 3 callback ports in addition to its main entry port. The initial rule set identified these 3 callbacks as additional services, but confusingly it named them the same name as the main entry port so that there appeared to be 3 additional services having the same name.

Another interesting anomaly came in the "invoke operation" abstractions; the initial rule set failed to identify the 6 locations where the top level service called operations on its 3 partners. It turned out that Travel Reservations used extensively the control flow elements of BPEL, leading to a much more complex program structure with more levels of nested XML. This structure defeated the simple initial rule.

Only 7 lines needed modification in the initial rule set to allow the system to handle all the Travel Reservations anomalies. The initial rule base correctly identified most abstractions, with only a few being missed due to anomalies in the way SOA artifacts are constructed. These results are very encouraging; only a few adjustments were need to improve the system's performance in accurately identifying abstractions, which might suggest that with every iteration of applying and refining the rules in the knowledge base, fewer and fewer changes are needed. This illustrates the adaptability of the rule-based approach and its suitability for the heterogeneous and changing nature of SOA applications.

VII. CONCLUSIONS

Ongoing maintenance of SOA composite applications will require scarce and expensive Software Engineering expertise. This expertise will be especially difficult to acquire and sustain because of the heterogeneity of SOA applications and the rapid changes to the environments in which they operate.

One approach to reducing this burden is knowledge-enhanced search: a search tool that integrates higher-level coaching about structures it can analyze with text-based matching for structures that it cannot. However, a search tool must go beyond a simple text matching engine on SOA artifacts because such artifacts require interpretation. An intelligent search tool must provide meaningful results that can assist a software maintainer to discover the relationships between components in the system. We developed a knowledge-based system that automates the task of interpreting SOA artifacts to generate useful abstractions on the collection of services and messages in a SOA composite application. The evaluation case study results indicate that a rule-based approach may provide the much needed adaptability that complex and heterogeneous SOA environment will impose on Software Engineering.

There are a number of enhancements that could be applied to the current tool including 1) a better user interface to provide a smooth integration of text search results and abstraction information and 2) integration of namespace rules to handle namespace information that occur in XML files of SOA artifacts. Ideally both the text search and the abstraction rules should take namespaces into account to improve both search precision and automated reasoning.

Researchers at several of our industry partners have suggested that search could be integrated with ontologies, both domain specific ontologies to clarify the terms used in a specific composite application, and Web Services ontologies to aid the novice in understanding the many element and attribute types that are defined in the standards. Ontologies could provide a deeper meaning to search results that could improve ordering and interpretation of output.

However, perhaps the most important research would be to try knowledge-enhanced search on a wider variety of SOA composite applications with different artifacts. It should be quite possible, for example, to develop rule sets for handling deployment descriptors, enterprise service bus configuration files, database definitions and possibly logged SOAP messages. Such research could help to define the benefits and limitations of knowledge-enhanced search and the application of rule-based systems to extract meaningful information from SOA artifacts.

ACKNOWLEDGMENT

Work described in this paper was partially supported by the University of West Florida Foundation under the Nystul Eminent Scholar Endowment and by the Blue Cross Blue Shield Association and by Intelligent Information Technologies, both industrial affiliates of the Security and Software Engineering Research Center (www.serc.net).

REFERENCES

[1] N. M. Josuttis, SOA in practice: The art of distributed systems design, O'Reilly, 2007, ISBN 0-596-52955-4.

[2] V. T. Rajlich, and K. H. Bennett, "A staged model for the software life cycle," Computer, vol.33, no.7, pp.66-71, Jul 2000, doi: 10.1109/2.869374.

[3] N. Gold, C. Knight, A. Mohan, M. Munro, "Understanding Service-Oriented Software." IEEE Software 2004; 21(2): 71-77. DOI: http://dx.doi.org/ 10.1109/MS.2004.1270766.

[4] G. Canfora and M. Di Penta, "New Frontiers of Reverse Engineering." Proceedings Future of Software Engineering. IEEE Computer Society:

Washington, DC, 2007; 326-341. DOI: http://dx.doi.org/10.1109/FOSE.2007.15.

[5] G. A. Lewis and D. B. Smith, "Service-Oriented Architecture and its implications for software maintenance and evolution." Proceedings Frontiers of Software Maintenance. IEEE Computer Society: Washington, DC, 2008; pp.1-10. DOI: http://dx.doi.org/ 10.1109/FOSM.2008.4659243.

[6] K. Kontogiannis, "Challenges and opportunities related to the design, deployment and operation of Web Services." Proceedings Frontiers of Software Maintenance. IEEE Computer Society: Washington, DC, 2008; 11-20. DOI: http://dx.doi.org/ 10.1109/FOSM.2008.4659244.

[7] M. P. Papazoglou, V. Andrikopoulos, and S. Benbernou, "Managing Evolving Services," IEEE Software, vol. 28, no. 3, pp. 49-55, May/June 2011, doi:10.1109/MS.2011.26.

[8] N. Gold and K. Bennett, "Program Comprehension for Web Services," Proc. 12th IEEE International Workshop on Program Comprehension 2004, p.151, doi:10.1109/ WPC.2004.1311057.

[9] I. D. Baxter, C. Pidgeon, and M. Mehlich. DMS: program transformations for practical scalable software evolution. In 26th International Conference on Software Engineering (ICSE 2004), 23-28 May 2004, Edinburgh, United Kingdom, pages 625–634, 2004.

[10] J. R. Cordy, T. R. Dean, A. J. Malton, and K. A. Schneider. Source transformation in software engineering using the TXL transformation system. Information & Software Technology, 44(13):827–837, 2002.

[11] B. Braun. SAVE - Static Analysis on Versioning Entities. Proceedings of SESS'08, May 17–18, 2008, Leipzig, Germany. pp 25 - 31.

[12] W. Zaghouani. RENAR: A Rule-Based Arabic Named Entity Recognition System. ACM Transactions on Asian Language Information Processing, 11(1), Article 2, 2012, pp 2:1 - 2:13.

[13] Y. Wang. Annotating and Recognising Named Entities in Clinical Notes. Proceedings of the ACL-IJCNLP 2009 Student Research Workshop, pages 18–26, Suntec, Singapore, 4 August 2009.

[14] E. Michelakis, R. Krishnamurthy, P. J. Haas, and S. Vaithyanathan. Uncertainty Management in RuleBased Information Extraction Systems. Proceedings of SIGMOD'09, June 29–July 2, 2009, Providence, Rhode Island, USA. pp 101 - 114.

[15] W. De Pauw, M. Lei, E. Pring, L. Villard, M. Arnold and J. F. Morar, "Web Services Navigator: Visualizing the execution of Web Services," IBM Systems Journal, vol. 44, no. 4, 2005, pp. 821-845, doi:10.1147/sj.444.0821.

[16] J. Coffey, L. White, N. Wilde, and S. Simmons, "Locating Software Features in a SOA Composite Application," Proc. 2010 Eighth IEEE European Conference on Web Services, ECOWS'10, pp. 99-106, doi:10.1109/ECOWS.2010.28.

[17] A. Yousefi and K. Sartipi, "Identifying distributed features in SOA by mining dynamic call trees," 27th IEEE International Conference on Software Maintenance, 2011, pp. 73-82, 2011 doi:10.1109/ICSM.2011.6080774.

[18] S. Halle, T. Bultan, G. Hughes, M. Alkhalaf and R. Villemaire, "Runtime Verification of Web Service Interface Contracts," Computer, vol. 43, no. 3, 2010, pp. 59-66, doi:10.1109/mc.2010.76.

[19] N. Wilde, J. Coffey, T. Reichherzer, L. White, "Open SOALab: Case Study Artifacts for SOA Research and Education," Principles of Engineering Service-Oriented Systems, PESOS 2012, Zurich, Switzerland, pp. 59-60, June 4, 2012, doi: 10.1109/PESOS.2012.6225941.

[20] "OASIS Web Services Business Process Execution Language (WSBPEL) TC", https://www.oasis-open.org/committees/wsbpel/, link accessed December 2012.

[21] "Web Services Description Language (WSDL) 1.1", www.w3.org/TR/wsdl, link accessed December 2012.

[22] E. R. Harold and W. S. Means, XML in a Nutshell, O'Reilly 2001, ISBN:0-596-00764-7.

[23] L. White, N. Wilde, T. Reichherzer, E. El-Sheikh, G. Goehring, A. Baskin, B. Hartmann, M. Manea, "Understanding Interoperable Systems: Challenges for the Maintenance of SOA Applications," 45th Hawaii International Conference on System Sciences (HICSS), pp. 2199-2206, 2012.

[24] T. Reichherzer, E. El-Sheikh, N. Wilde, L. White, J. Coffey, and S. Simmons , "Towards intelligent search support for web services evolution: identifying the right abstractions," 13th IEEE International Symposium on Web Systems Evolution (WSE-2011), pp.53-58, 30 Sept. 2011, doi: 10.1109/WSE.2011.6081819.

[25] "Drools - The Business Logic integration Platform", http://www.jboss.org/drools, link accessed December 2012.

[26] "Apache Lucene - Apache Solr," http://lucene.apache.org/solr/, link accessed December 2012.

AUTHORS PROFILE

George Goehring received his M.Sc. in Computer Science from the University of West Florida in 2012. His graduate research focused on providing knowledge-based enhancement of SOA composite applications through the use of artificial intelligence techniques. His research interests include distributed systems and artificial intelligence.

Dr. Thomas Reichherzer is an Assistant Professor in the Computer Science Department at the University of West Florida. He received a Ph.D. in Computer Science at Indiana University in 2009. Dr. Reichherzer's interest include knowledge representation, the Semantic Web, information retrieval, management, and visualization. More recently, he focused on sentiment analysis of unstructured text and AI approaches to smart home environments.

Dr. Eman El-Sheikh is the Associate Dean for the College of Arts and Sciences and an Associate Professor of Computer Science at the University of West Florida. She received her Ph.D. and M.Sc. in Computer Science from Michigan State University and B.Sc. in Computer Science from the American University in Cairo. Her research interests include artificial intelligence-based techniques and tools for education, including the development of intelligent tutoring systems and adaptive learning tools, agent-based architectures, knowledge-based systems, machine learning, intelligent support for service-oriented architectures, and computer science education.

Dr. Dallas Snider is an Assistant Professor of Computer Science at the University of West Florida. He received his Ph.D. in Integrated Computing and M.S. in Instrumental Sciences from the University of Arkansas at Little Rock. He received a B.A. in Physics from Hendrix College. Dr. Snider's teaching and research interests include data mining, data warehousing.

Dr. Norman Wilde is Nystul Chair and Professor of Computer Science at the University of West Florida. He received his BS (Hons) in Physics from the University of Manitoba in 1967 and his Ph.D. in Mathematics and Operations Research from the Massachusetts Institute of Technology in 1971. Dr. Wilde has been on the faculty of the University of West Florida since 1986. His research interests are Software Engineering, Software Maintenance, Program Comprehension, and Services Oriented Architecture.

Dr. Sikha Bagui is Chair and Associate Professor in the Department of Computer Science at the University of West Florida, Pensacola, Florida, USA. Dr. Bagui's primary research areas are database design, web databases, data mining and statistical computing. Dr. Bagui has published many journal articles and co-authored several books.

Dr. John W. Coffey holds a B.S. in Psychology from the College of William and Mary (1971), a B.S. in Systems Science (1989), an M.S. in Computer Science/Software Engineering (1992), and an Ed.D. with an emphasis in Computer Science (2000) from the University of West Florida (UWF). He was one of the first members of the Institute for Human and Machine Cognition (IHMC) and he has worked with that organization for many years. He has been in the Department of Computer Science at the University of West Florida since 1992, starting as a Lecturer and working his way up to his current rank of Professor. He has published a total of more than 70 refereed journal articles, book chapters, and conference proceedings. His research interests include advanced technology for education, knowledge elicitation and representation, student modeling, web services, and Service Oriented Architecture.

Dr. Laura J. White is an Associate Professor in the Department of Computer Science at the University of West Florida, and a Visiting Research Scientist at the Florida Institute for Human & Machine Cognition. She received her B.S. from the University of New Mexico, M.S. from the Naval Postgraduate School, and her Ph.D. from Capella University. Laura is a retired U.S. Navy Surface Warfare Officer, and her research interests include software engineering teams, software engineering maintenance, mobile programming, and the scholarship of teaching and learning.