Service-Oriented Architecture: Introducing a Query Language

Sebastian Günther, Claus Rautenstrauch, Niko Zenker

Otto-von-Guericke-Universität Magdeburg
School of Computer Science, Business Informatics Research Group
PO-Box 4120
39106 Magdeburg, Germany
sebastian.guenther@iti.cs.uni-magdeburg.de
claus.rautenstrauch@iti.cs.uni-magdeburg.de
niko.zenker@iti.cs.uni-magdeburg.de

Abstract: Language offers human beings the ability to exchange information. Whether the information is understood or not depends on the structure, complexity and knowledge of the language. Language is designed for a certain context. In this paper, the context is Service Oriented Architectures. We present a query language for retrieving information about the resource consumption of services. The language includes major concepts and resource types, features an easy to understand syntax and is powerful enough to satisfy the information needs of customers and providers of services.

1 The need for a query language

In modern computing environments, Service Oriented Architectures (SOA) plays a major role [26] [31]. In order to engage the two forces of heterogeneity and change, SOA allows to make components for single services and to integrate them into the IT infrastructure of a company [10]. SOA can be implemented in numerous ways, the important ones being Web Services [24] [12] and the Enterprise Service Bus [7].

From the technical view, SOA are Information Technology (IT) assets with coarse-grained interfaces and a separate technical implementation [22]. Thus, they can be considered components according to the object oriented programming paradigm [18]. While each component has an interface to query for data from a service [22], the real implementation uses software components. As software, the components use computational resources like CPU, RAM or disk space [12]. Normally, the components run inside an integrated architecture, which as a whole consumes resources. With detailed measurements, a number of benefits, like a higher utilization, can be achieved. However, there is yet no standardized way to measure these resources.

This paper proposes a query language for stating the resource consumption of SOA. Chapter 2 explains the term Service Oriented Architecture. In the next chapter 3 the

resources to be measured are proposed. Section 4.1 explains the requirements of the language, both from a total point of view and from the view of the two stakeholder customer and provider. This is followed by an excursus to the construction of computer languages. We specify the semantics and syntax of the language in section 4.3. Chapter 5 gives an overview to the benefits of the query language and concludes the paper with presenting open research issues. The authors only use the scientific method of deduction (see [23] for an overview) and thus no empirical analysis to produce the research results.

2 Service Oriented Architectures

2.1 Evolution of computer systems design

Today's IT architectures are a complex system of interacting subsystems. They are implemented in heterogeneous programming languages, they offer different data and functionality, and are integrated with heterogeneous technologies in the whole enterprise architecture [19]. There are three main layers of a system that contribute to the architecture [13]. In the *data layer*, all data that must be persistently stored is kept. This storage could be a simple file or a database. The *logic layer* defines the domain concepts of the system [18]. In the case of an online shop, such concepts would be the shopping cart, a product or a customer. Entities of the real world are modeled as objects and data types. Finally, the *presentation layer* produces the user interface. For the online shop example again, this would be HTML[1] code that is presented in the browser. The semantics of these three layers can be found often in literature, although the names are differencing, cf. [18] [15] [13] [11].

In the history, these layers were either strongly related or separated from each other. In monolithic applications, In monolithic applications, all layers are tightly coupled. The thick client separated data and presentation. Finally, in classic three-tier architecture, all layers were separated [15]. This also roughly corresponds to the technical development of computer systems, starting from the "Batch Era" to the "Terminal/Host Era" and the "Client/Server Era" [15] – the design evolution lead to a decoupling of layers. Specialized applications might even require more than three layers, for example J2EE[2] poses the layers Client, Presentation, Business, Integration and Resource [13]. In typical web applications, the so called Model-View-Controller layering scheme is used [30]. These layers encapsulate main responsibilities and dependencies. Most modern systems are constantly requested to change. The changes could come from market conditions, like a competition between companies, or the customizing of products for customers [7] [26]. The changes can be treated inside the layers separately and thus support flexibility.

[1]Hyper Text Markup Language
[2]Java 2 Platform Enterprise Edition

2.2 Implications of SOA for the industry

In a recent study from HP [31], managers of companies were asked to present their thoughts about SOA. The five major awaited benefits are:

- Improve overall end-to-end business process flexibility and performance
- Improve IT flexibility/adaptability via rapid application and business process creation/modification
- Improve system interoperability and reduce integration costs
- Deploy new applications and business services we could not otherwise afford
- Better match cost of software to use and business value

Chappell additionally identifies in another study two important benefits for the integration of systems: The infrastructure replacement/upgrade and IT cost cutting [7]. Those awaited benefits show the importance of SOA.

With the measurement of the "application up-time and availability" [31] the contribution of a service to the business is shown. These services/applications directly correspond to either electronic business processes, like processing a shopping cart or financial transactions [26], or to real world processes, like the exchange of foreign currencies or withdrawal of money [22]. These services form major parts of the SOA.

2.3 Technological architectures

Web Services (WS) [24] and the Enterprise Service Bus (ESB) [7] are often used technologies to implement a SOA. Both the ESB and WS use standards. ESB defines the information format, the messaging and for the handling of messages. It is responsible for the uniform transformation and routing of all messages from and to different systems. For the connection to the ESB, the system has to have an adapter which translates the message [7]. With this approach, a multitude of systems can be connected.

The standard for WS is XML as the message-format, WSDL[3] for the description of Web Services and a uniform repository for the services called UDDI[4] [25]. The protocol SOAP[5] is used as the platform for service communications [27]. For the transportation of data, among many other possible protocols, HTTP[6] and SMTP[7] are allowed [16]. The client looks for a service at the registry and finds one that is appropriate. In their basic idea, both WS and ESB use a standard format for the communication through messages. The differences are in the standard: The ESB can use a proprietary format, although XML evolved to the de-facto standard [7] while in WS, XML data using the WSDL is

[3]Web Service Description Language, see http://www.w3.org/TR/wsdl
[4]Universal Description, Discovery and Integration, see http://uddi.org/pubs/uddi_v3.htm
[5]Simple Object Access Protocol, see http://www.w3.org/TR/soap12-part1/
[6]Hypertext Transfer Protocol
[7]Simple Mail Transfer Protocol

exchanged [16]. In ESB, different systems can be connected via the adapter that translates messages; in WS the services must use WSDL.

For building SOA applications, the ESB also implements a workflow and message system. With WS, only the message exchange and its formats are specified. If a company would like to implement a SOA with WS, it would need to install an additional workflow system to handle the service requests.

2.4 A marketplace of services

The offered and requested services can form a marketplace. In principle there are two different parties: The *providers* and *customers*. The provider are differentiated into the *service provider* who offer the above specified services, and the *resource provider* who host an hardware infrastructure on which the services run. These two groups can be the same if the service provider also has enough computing power to host the services as well.

The customer wants to query for available services and computing power, which is described by the following basic process (note that the basic ideas for this process stem from [5]). First he selects among the service repository a service that satisfies the customer's requirements. He then has both a string representation of the service and also information about the resource consumption of the service, e.g. which CPU clock speed is the minimum and how much RAM the service uses. With this information, the customer queries the repository of resource providers. He gets a list with the description of the available amount of resources. Finally the customer selects a service and a resource provider, all parties sign a contract, and the results of the service computation are returned to the customer.

3 Resource measurement

Resources can be separated into five types: human resources, application systems, technical infrastructure, facilities, and data [14]. In this paper, access to information about the application systems and technical infrastructures resources are important. In [33] a limited list of computational resources such as CPU Time (1 hours MIPS), disk space (GB per month), or disk performance (MB per second) can be found. However, these resources only have value as initial ideas and needs to be completed by other resources as well. Thus, the following resource index is proposed[8]. Each resource object is additional specified with its attributes and the attributes data type in a Java-like syntax.

RAM

```
String Name = "RAM"
String Type = "RDRAM" | "SDRAM"
Int Size = 1024
```

[8] The proposed list is just an excerpt of the whole list, containing for example: RAM, SWAP, HD, CPU, Jobs, Network, and OS. For brevity, they are left out here.

```
String Unit = "B" | "MB" | "GB"
```

CPU

```
String Name = "CPU"
String Type = "Pentium4" | "AMD64-X2"
Float GHZ = 4.2
Int CacheSize = 256
Int NumberOfCores = 2
Int CurrentCore = 0
```

OS

```
String Name = "OS"
String Type = "WindowsXP" | "Linux"
```

4 SOA query language

4.1 Requirements

The query language at its core provides a channel to satisfy the information needs of providers and customer. Like stated in 2.4, customer will query providers for their available resources. On the other side, the providers could use the same information to collect real-time data about the resource consumption of certain services. Both groups have different goals and requirements for the queries. We first want to formulate the overall requirements (partly stemming from [20]) and desirable implementation characteristics of the language. Then we analyze in detail which information needs providers and customers have. The presented information needs are, among others, form the basic requirements of the SOA query language (SOA QL). The outlook shows some additional needs in 5.2.

Overall requirements

Easy Syntax The language should be easy to read and to be used by humans.

Expressiveness The query language is used by different stake holders and must response to different resources.

Machine independence The language is not restricted to specific computer architecture in order to be executed.

Extensibility When new concepts or resources to be measured are introduced, the language can be easily adapted.

Implementation characteristics

- The language is a domain specific language (for more details see [21]).

- Expressions in the language can be embedded inside other languages, like embedded SQL statements [19], to integrate using SOA-QL in application development

Providers have the following information needs.

- List resources of a server
- Show available and used resources
- List all running services on a server
- List historical data about the resource consumption (hours, days, time frames)

Customers want to have the following information.

- List all servers and their available resources
- List the resource consumption of own running services
- List servers that offer a specified services

4.2 Constructing a language

Language is a unique human endowment. With language, humans can exchange their ideas and thoughts [8]. Opposed to the natural languages of humans, languages in computer science have to be formalized in order to be used for programming. The main parts of a language description are syntax and semantics [20] - they are relevant for a language to be "specified without ambiguity" [28].

When looking into papers about programming languages, syntax and especially semantics are not defined explicitly and used rather opaque. [4] describes semantics as the transformation to Java classes. [9] uses semantics to describe a "functions from continuous 2D space to colors", and [2] describes concepts. [15] separates semantics into translational (statements of an domain-specific language are transformed to statements of another language with their semantics) and trace-based (output of a program can be analyzed, e.g. with a parser, and "an execution trace language is constructed and then mapped to expressions in the language whose semantics is being defined"). While language-semantics describe the constructs of a programming language, we also need to specify the underlying ontological concepts of a language as well - the *concept-semantics*. The same multitude is in the definition of syntax. In [15], for the syntax either the BNF[9] or an UML-based[10] meta-model is used. [2] uses description logic for the syntax and semantics. [32] uses a syntax that resembles common programming languages; [9] uses a host language for the description, so that the syntax is derived

To be clear, a definition of the concept-semantics and syntax is found in the following paragraph. In the following, the word semantics means concept-semantics..

[9]Backus Naur Form - a formal method to describe the grammar of an language
[10]Unified Modelling Language - a set of graphical design methods to describe the structure and architecture of an computer program

Concept-semantics Concept-semantics describes the meaning of concepts of a language. Concepts are formed through the conceptualization of things, seen top-down or domain specific. Concepts can have attributes or relationships to each other, like being a sub concept or a compound object. Concepts and their relationships form the vocabulary of the language [19]. The complete vocabulary is described in ontology. Thus, the semantics can be captured inside an ontology as the repository of "knowledge representation in that domain" [6].

Syntax Describes the grammatical form of sentences. Sentences consist of the concepts and other grammatical word types. In programming languages, they are often called tokens [20]. The relationships of the concept can also be modeled with the syntax, so that the meaning of the sentence can be easily captured by the reader.

4.3 Constructing the SOA QL

4.3.1 Overview

The before mentioned information culminates now into the construction of the query language. We use the EBNF for the syntax description and prose text for the semantics.

4.3.2 Syntax

This section lists the syntax definition of the SOA-QL. The Extended Backus-Naur Form (EBNF) is used to construct a context-free grammar (see e.g. [28]). For completeness, here is a short summary of the basic notations.

() = Grouping of expressions
| = Alternative terms
[ ] = Optional terms
{} = Recursion from 0 to *

This EBNF specification contains basis formatting for parentheses, commas and the keyword "END" used in the examples above. Formatting for white space is not given, but it should be used for better readability. Terminals for time variables are considered discrete and modeled as concrete values. For brevity, we make two compromises: The detailed specification of the syntax for resource listening is omitted (the example of the definition for RAM should be sufficient) and the sequence "..." denotes subsequent characters or number. Also, the syntax for the response queries is not specified.

```
<query> ::= <service> | <server> | <measure>;
<service> ::= 'SERVICE' 'SERVER' {<id>}
  ['RESOURCE' {<resources>}]
  ['TYPE' <type>]
  'END';
<server> ::= 'SERVER' 'SERVICE' {<id>}
  ['RESOURCE' {<resources>}]
  ['TYPE' <type>]
  'END';
<measure> ::= 'MEASURE' ('SERVER' {<id>} |
```

```
  'SERVICE' {<id>})
  'RESOURCE' '('{<resource-names>}')'
  'TYPE' <type>
  ['TIME' <time>]
  'END';
<id> ::= {<identifier> | ,};
<resource-names> ::= 'RAM' | 'SWAP' | 'HD' | 'CPU' |
  'Jobs' | 'Network' | 'OS';
<time> ::= <time-period>;
<type> ::= 'Max' | 'Used' | 'Free';
<resources> ::= (
   '(' ( 'Name' '=' "RAM"','
  'Type' '=' ("RDRAM" | "SDRAM)"','
  'Size' '=' {<number>}','
  'Unit' '=' ("B" | "MB" | "GB") )
  | ... | ')'
  ) | '*';
<time-period> ::= <minutes>'M' | <hours>'H' |
  <days>'D' | <month>'.'<day>'.'<year> '-'
  <month>'.'<day>'.'<year>;
<identifier> ::= 'a' | 'b' | ... 'z' | 'A' | 'B' |
... 'Z' | '.' | '?' | '!' | '+' | '#' | '*';
<minutes> ::= '1' | '2' | ... | '59';
<hours> ::= '1' | '2' | ... | '23';
<month> ::= '1' | '2' | ... '12';
<days> ::= '1' | '2' | ... | '365';
<day> ::= '1' | '2' | ... | '31';
<year> ::= '2000' | '2001' | ... | '2100';
<number> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';
```

4.3.3 Semantics

When considering the requirements, a total number of three different queries can be found: List all resources, list all services, and list all available servers. These basic types can be enhanced in their search, like specifying only servers that offer a certain service in combination with a certain amount of RAM size. In the following, an example for each of these queries is constructed. The important part of semantics were already defined in chapter 3 (the resources) and in section 2.4 (the stakeholders).

A *measure request* is a simple query to list the available resources. The user either queries a server or a service for a list of resources. The type can have three values: *Max* lists the installed hardware basis, *Free* lists the still available resources and *Used* shows the currently used resources.

```
MEASURE
  SERVICE Service.Business.Financial.CheckAccount
  RESOURCES (CPU, RAM, HD)
  TYPE Max
END
```

This assumes that all services of the type “Service.Business.Financial.CheckAccount” on all servers are queried. The answer to the query may look as follows. Note that the return values and their types were defined in chapter 3.

```
MEASURED
  SERVER server3.soacluster.com
  RESOURCE
    (Name = "CPU", Type = "Pentium4-M", GHZ = 4.2,
```

```
      CacheSize = 256)
    (Name = "RAM", Type = "RDRAM", Size = 2048,
      Unit = "MB")
  END
END
```

The *service request* lists all services that run on a server. It basically has the following form:

```
SERVICE
  SERVER server3.soacluster.com
END
```

A *server request* will output all available servers on which services can be hosted. The example includes the specification of a service and will output only relevant servers.

```
SERVER
  SERVICE Service.Business.Shopping.ShoppingCart
END
```

Additionally to the resources, clients may wish to query the resource consumption for a specific time-frame, like hours or days.

```
MEASURE
  SERVER server3.soacluster.com,
         server7.soacluster.com
  SERVICE Service.Business.Financial.CheckAccount
  RESOURCES *
  TIME 3H
END
```

The return values would have the form of a list, each entry containing the name of the server and the average resource consumption for the specified time frame.

```
…
MEASURED
  SERVER server7.soacluster.com
  SERVICE Service.Business.Financial.CheckAccount
  RESOURCE (Name = CPU, GHZ = "0.7"),
    (Name = "RAM", Size = 760, Unit = "MB"))
END
…
```

Considerations about some special cases for the queries end this section. The three different queries measure, server and service can be combined. A measure query already includes functionality of filtering for servers and service; it scales to the case of selecting a single service on a single server. On the other side, service and server queries can be enhanced by listing the resources - combined with the type argument the user can get a list of services that offer a specified amount of free resources. If different servers are to be queried, they are added to list. In the case that a server has multiple parts of hardware, for example CPU, the answer will simply contain the different instances. The query can also contain a specified type and time - the type will specify the meaning of the given time frame for either max, free or used resources.

5 Outlook and further research

5.1 Open issues

The language still offers a lot of potential for evolution in the future. The suggested naming mechanism for servers and service has to be specified, for example with the conventions derived from ontology and semantic web technologies like in [3]. When thinking about implementation, the language also needs a parsing mechanism. If a query is parsed, errors like a misspelled resource have to be considered. Also a specification for the return types and a full specification of the language including all resources is needed.

5.2 Enhancements of the query language

Once the provider and service is found, the user wishes to deploy the service. To provide that, negotiation capabilities must be added. With this, a portal where customers and providers offer their needs can be designed - the portal often forms the basics of an electronic marketplace [1]. If the result offers a whole collection of providers for the same task, one service could be selected that fits the circumstances of the customer. The customer should thus be able to include his quality needs for the execution of the service and his further requirements. Also, the information needs for customers and providers can be extended in the future, as follows.

For providers

- Show service belonging to a certain customer
- Show service of a certain service provider
- Calculate pricing information for each service

For customers

- Contract negotiation
- Auction model for service usage pricing [5]
- Compliance to Service Level Agreements [29]

5.3 Theory of the construction of domain specific languages

One of the authors targets to create a theory for the construction of computer languages. The first goal is to identify the needed specifications of a language. In this paper, the syntax and semantics of the SOA QL were described. But also other parts of the language may to be considered, e.g. pragmatics describe the usage of a language by its speakers [17]. The usage could change, new terms and concepts could be introduced to

the language, old words and concepts could vanish. The specification will need a mechanism to deal with these changes, the language and its specification need to be adapted. It is also interesting to review the different formats for specification and to measure their suitability in describing a language. It is very important to offer an executable language, e.g. with a self written parser or by formulating the language inside another language that already is executable. These ideas, among other, will be formulated in a theory. So the SOA QL is one of the application areas for this theory - and it can be tested in the practice right from the start.

References

[1] Blunck, bpu, Increaon, and P. C. Veltins, editors. Dynamic Alliance Auctions: A Mechanism for Internet-Based Transportation Markets. Schäffer-Poeschel Verlag für Wirtschaft. Steuern, Recht GmbH & Co. KG, Stuttgart, 2003.

[2] A. Borgida. Description logics in data management. IEEE Transactions on Knowledge and Data Engineering, 7(5):671#682, 1995.

[3] J. Broekstra, M. Klein, S. Decker, D. Fensel, F. van Harmelen, and I. Horrocks. Enabling knowledge representation on the Web by extending RDF Schema. Computer Networks, 39(5):609#634, 2002.

[4] A. Bryant, A. Catton, K. De Volder, and G. Murphy. Explicit programming. Proceedings of the 1st international conference on Aspect-oriented software development, pages 10#18, 2002.

[5] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in Grid computing. Concurrency and Computation: Practice and Experience, 14(13-15):1507#1542, 2002.

[6] B. Chandrasekaran, J. Josephson, and V. Benjamins. The Ontology of Tasks and Methods. Proceedings of the 11th Ban# Knowledge Acquisition for Knowledge-Based System Workshop (KAW98), pages 18#23, 1998.

[7] D. Chappell. Enterprise Service Bus. O'Reilly, 2004.

[8] N. Chomsky. Language and Mind. Harcourt Brace Jovanovich, Inc., New York, Chicago et. al., 1972.

[9] C. ELLIOTT, S. FINNE, and O. DE MOOR. Compiling embedded languages. Journal of Functional Programming, 13(03):455#481, 2003.

[10] M. Endrei, J. Ang, A. Arsanjani, S. Chua, P. Comte, P. Kroghdahi, M. Luo, and T. Newling. Patterns: Service-Oriented Architecture and Web Services. IBM Corp., International Technical Support Organization, IBM redbooks edition, 2004.

[11] M. Fayad, D. Schmidt, and R. Johnson. Building application frameworks: objectoriented foundations of framework design. John Wiley & Sons, 1999.

[12] M. Flehming. Design an Integraton of SLA Monitoring Negotiation Capabilities. http://tb0.asg-platform.org/download/downloadrequest.php?asgdocument=D5.II-7.pdf, retrieved on 20th March 2007, 2006.

[13] M. Fowler. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002.

[14] W. Goltsche. COBIT kompakt und verständlich. Vieweg, 2006.

[15] J. Greenfield, K. Short, S. Cook, and S. Kent. Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools. Wiley Publishing, Inc.,

2004.
[16] J. Hasan. Service Oriented Architecture in C#: Using the Web Services Enhancements 2.0. Apress, 2004.
[17] F. v. Kutschera. Sprachphilosophie. Wilhelm Fink Verlag, München, 1975.
[18] C. Larmann. Applying Uml and Patterns: An Introduction to Object-Oriented Analysis and Design and the Uni#ed Process. Prentice Hall PTR, 2nd edition, 2002.
[19] U. Leser and F. Naumann. Informationsintegration - Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen. dpunkt Verlag, Heidelberg, 2007.
[20] K. Louden. Programmiersprachen: Grundlagen, Konzepte, Entwurf. International Thomson Publishing GmbH, 1994.
[21] M. Mernik, J. Heering, and A. Sloane. When and how to develop domain-specific languages. Report Software Engineering, SEN-E0309, 2003.
[22] E. Newcomer and G. Lomow. Understanding SOA with Web Services. Addison-Wesley Professional, 2004.
[23] B. Niehaves. Epistemological Perspectives on Multi-Method Information Systems Research. Proceedings of the 13th European Conference on Information Systems (ECIS 2005), 2005.
[24] P. Offermann, C. Schröpfer, and M. Schönherr. Entwurf eines Enterprise Architecture Framework für serviceorientierte Architekturen - Betrachtung am Beispiel einer erweiterten UN/CEFACT Modeling Methodology. E-Organisation: Service-, Prozess-, Market-Engineering. 8. Internationale Tagung Wirtschaftsinformatik, Band 1, 2007.
[25] E. Ort. Service-Oriented Architecture and Web Services: Concepts, Technologies, and Tools. Sun Developer Network, 2005.
[26] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing Research Roadmap. Service Oriented Computing (SOC), 5462, 2006.
[27] M. Papazoglou and W. van den Heuvel. Service-oriented computing: state-oft-he-art and open research issues. Telematica Instituut, 2003.
[28] M. L. Scott. Programming Languages Pragmatics. Morgan Kaufmann Publishers, 2000.
[29] R. Sturm, W. Morris, and M. Jander. Foundations of service level management. Sams, Indianapolis, 2000.
[30] D. Thomas and D. H. Hansson. Agile Web Development with Rails: A Pragmatic Guide. The Pragmatic Programmers LLC, 2005.
[31] M. J. Turner. HP's adaptive Infrastructure powers Enterprise Business Priorities. HP's Whitepaper, 2006.
[32] M. Van Den Brand, J. Heering, P. Klint, and P. Olivier. Compiling language definitions: the ASF+ SDF compiler. ACM Transactions on Programming Languages and Systems (TOPLAS), 24(4):334-368, 2002.
[33] R. Zarnekow, W. Brenner, and U. Pilgram. Integrated Information Management - Applying Successful Industrial Concepts in IT. Springer Berlin Heidelberg, 2006.