



Provided by the author(s) and University of Galway in accordance with publisher policies. Please cite the published version when available.

Title	SESA: Emerging Technology for Service-Centric Environments
Author(s)	Vitvar, Tomas; Zaremba, Michal; Zaremba, Maciej; Moran, Matthew; Fensel, Dieter
Publication Date	2007
Publication Information	Tomas Vitvar, Michal Zaremba, Maciej Zaremba, Matthew Moran, Dieter Fensel "SESA: Emerging Technology for Service-Centric Environments", IEEE Software, 24(6), 2007.
Publisher	Institute of Electrical and Electronics Engineers
Link to publisher's version	doi: doi:doi: 10.1109/MS.2007.178
Item record	http://hdl.handle.net/10379/444
DOI	http://dx.doi.org/10.1109/MS.2007.178

Downloaded 2024-04-23T22:20:36Z

Some rights reserved. For more information, please see the item record link above.





SESA: Emerging Technology for Service-Centric Environments

Tomas Vitvar, Michal Zaremba, Matthew Moran, Maciej Zaremba, and Dieter Fensel

Vol. 24, No. 6
November/December 2007

This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.

IEEE  computer society

© 2007 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

For more information, please see www.ieee.org/web/publications/rights/index.html.

SESA: Emerging Technology for Service-Centric Environments

Tomas Vitvar, Michal Zaremba, Matthew Moran, Maciej Zaremba, and Dieter Fensel, *Digital Enterprise Research Institute*

Extending service-oriented architectures with semantics can help create service-centric information systems that better adapt to changes throughout software systems' lifetime.

Service-oriented architectures—and particularly the Web service technologies that enable them, such as WSDL (Web Services Description Language) and SOAP—are widely acknowledged for their potential to revolutionize computing. However, existing SOAs will prove difficult to scale without a proper degree of automation. SOAs' success depends on resolving fundamental challenges that existing SOA technologies don't sufficiently address: search, integration, and mediation.

In large-scale, open, service-centric environments, thousands of services will have to be discovered, adapted, and orchestrated on the basis of user needs. Because SOA technologies employ only XML descriptions, they offer only manual support for integration that usually operates on a rigid configuration of workflows or services. Although XML is flexible and extensible, it defines only the data's structure and syntax.

Extending SOAs with semantics offers scalable, adaptive automation. The goal is to design a *semantically enabled SOA* (SESA) and a technology promoting on-the-fly personalization and adaptability of business requirements. The SESA adopts a service model that uses semantics for rich description of both the services offered and the capabilities that potential users require. This model defines essential functionalities for the dynamic integration of services. The SESA aims to fulfil users' goals through logical reasoning over semantic descriptions.

Ultimately, users won't need to be concerned with the processing logic; they will be able to focus just on the result and its quality.

Governing principles

The SESA enables an open, service-centric environment where service orientation, intelligence, and seamless integration are the key to providing services to their users. Three principles drive SESA research, design, and implementation:

- The *service-oriented principle* promotes service reusability, loose coupling, abstraction, composability, autonomy, and discoverability.
- The *semantic principle* allows a rich description of information and behavioral models that enables automation through logical reasoning in service discovery, negotiation, selection, mediation, and so on.
- The *problem-solving principle* underpins

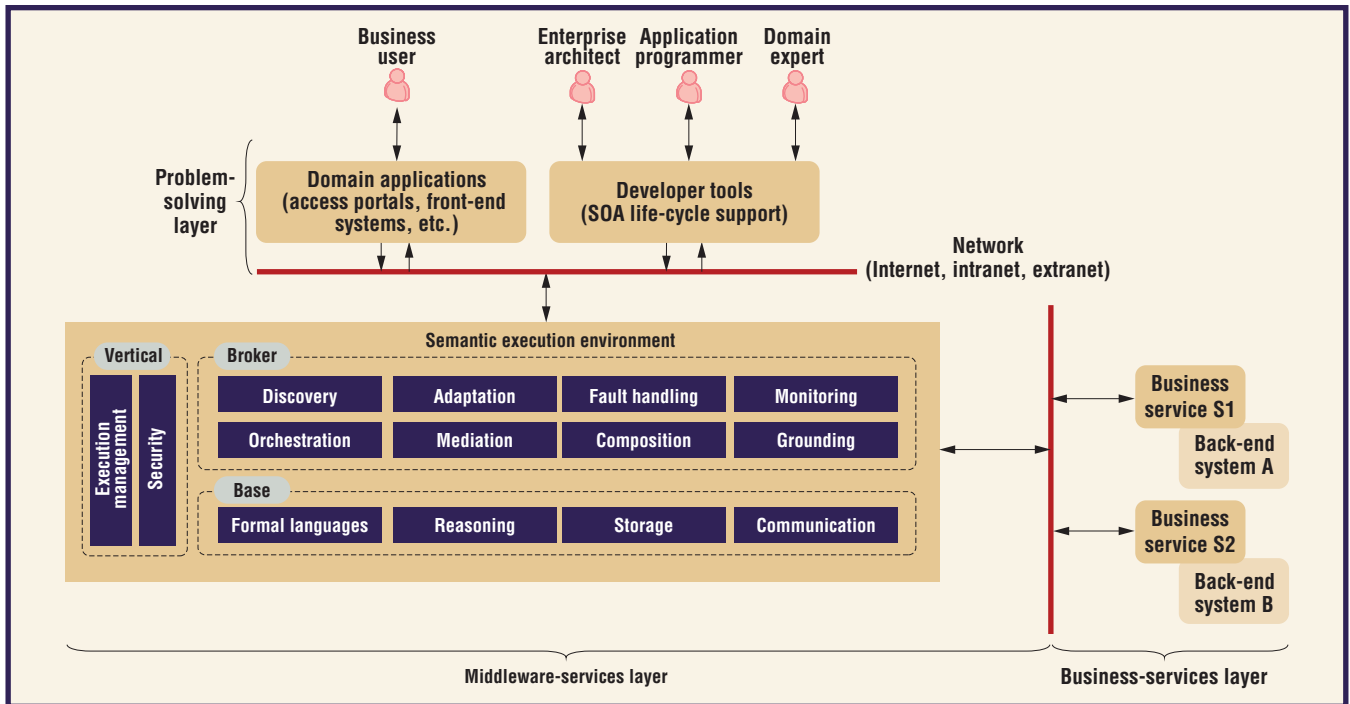


Figure 1. The semantically enabled service-oriented architecture (SESA).

the architecture's ultimate objective: goal-based discovery and invocation of services. Users (service requesters) describe requests as goals semantically and independently from services, while the architecture solves those goals through logical reasoning over their descriptions, as we mentioned before.

The SESA and its related technology

The Web Service Modeling Ontology provides a conceptual model for describing domain ontologies, user goals, services, and mediators.¹ To formally describe these elements, the Web Service Modeling Language defines several complementary language variants allowing for the logical expressiveness of both description logic and logic programming.¹ WSMO and WSMML provide the basis for a semantic technology that's well suited for the SESA and its underlying principles. (For a comparison of SESA technology with other approaches, see the "Related Work on Adding Semantics to Web Services" sidebar).

SESA distinguishes two types of services. *Middleware services* are the main facilitators for searching, integrating, and mediating *business services*. The business services are ex-

posed by the service providers' back-end systems, which are integrated within the SESA. Through the functionality of these types of services, the SESA aims to support

- the business users who consume the functionality of business services through some domain applications, and
- the engineers (enterprise architects, application programmers, and domain experts) who consume middleware services to model, deploy, assemble, manage, and maintain business processes throughout the SOA life cycle.

As figure 1 illustrates, the SESA has three main layers: the middleware-services layer, the business-services layer, and the problem-solving layer. The WSMO conceptual model and its elements of ontologies, services, and goals map directly to the SESA business-services and problem-solving layers, while mediators map to the middleware-services layer.

The middleware-services layer

The middleware services reside in the *semantic execution environment*. They operate mainly on the semantic-service model of business services, aiming to facilitate the seamless integration of business services. The OASIS (Organization for the

Related Work on Adding Semantics to Web Services

Besides the Web Service Modeling Ontology (see the section “The SESA and its related technology” in the main article), the OWL-S¹ ontology can be used for semantic description of services. OWL-S comprises three subontologies:

- *ServiceProfile* describes a Web service’s functionality.
- *ServiceModel* defines a Web service’s behavior.
- *ServiceGrounding* links *ServiceModel* and the description of the concrete Web service that the Web Services Description Language (WSDL) provides.

OWL-S uses the Web Ontology Language (OWL) for descriptions of its elements. OWL currently doesn’t support rule languages, which are essential for describing functional and behavioral service semantics using logical conditions. In addition, OWL-S doesn’t include the user request in its conceptual model, and it doesn’t assume that service environments are inherently heterogeneous.

To overcome these deficiencies, WSMO introduces a proper language layering, including logic-programming languages suitable for describing service semantics using logical conditions. It also introduces *ontological role separation*, clearly distinguishing between the requester and the provider side (that is, between goals and services). Plus, it introduces *mediators* as the core of a conceptual model treating heterogeneity as a natural aspect of Web services.

In addition, some technologies are building on the OWL-S model. Among others, the OWL-S Virtual Machine provides a general-purpose client for the invocation of Web services based on OWL-S descriptions.² Although the OWL-S VM proves the OWL-S model’s value, it addresses only a subset of the functionality that a *semantically enabled service-oriented architecture* (SESA) offers. For example, the OWL-S VM doesn’t directly address mediation and lacks goal-driven execution.

Other specifications also aim to add semantics to services. The Semantic Annotations for WSDL and XML Schema (SAWSDL)³ is a recent World Wide Web Consortium (W3C) project that adopts WSDL-S⁴ specifications. SAWSDL aims to define simple extensions to annotate various WSDL ele-

ments with semantic descriptions. While SAWSDL doesn’t define any semantics specifically for service descriptions, it can serve as a grounding mechanism between WSMO semantics and WSDL elements.⁵

Based on WSDL-S, METEOR-S (Managing End-to-End Operations with Semantics, <http://lsdis.cs.uga.edu/projects/past/METEOR>) defines a complete service life cycle based on service semantics.⁴ It lets you create WSDL-S descriptions from annotated source code, automatically publish WSDL-S descriptions in enhanced UDDI registries, and generate OWL-S descriptions from WSDL-S for grounding.

In addition, you can view the SESA as complementing the W3C’s Web Services Architecture,⁶ building on its principles. However, some differences exist. For example, the SESA includes semantic mediation, refines the Web Services Architecture goal state to a WSMO goal, and includes WSMO choreography interfaces for both goals and services (a WSMO choreography describes a service’s external behavior). For conceptual models of Web services specifications (WS-*) such as policy, security, or reliability, the SESA either adopts these specifications as part of the business service model or adopts them through the underlying Web service technology.

References

1. D. Martin et al., “Bringing Semantics to Web Services: The OWL-S Approach,” *Proc. 1st Int’l Workshop Semantic Web Services and Web Process Composition (SWSWPC 04)*, Springer, 2004, pp. 26–42; www.springerlink.com/content/rl5r1c8v64xvf0r8.
2. M. Paolucci et al., “The DAML-S Virtual Machine,” *The Semantic Web—ISWC 2003*, LNCS 2870, Springer, 2003, pp. 290–305.
3. J. Kopecký et al., “SAWSDL: Semantic Annotations for WSDL and XML Schema,” to be published in *IEEE Internet Computing*, vol. 11, no. 6, Nov./Dec. 2007.
4. A. Patil et al., “METEOR-S Web Service Annotation Framework,” *Proc. 13th Int’l Conf. World Wide Web (WWW 04)*, ACM Press, 2004, pp. 553–562.
5. T. Vitvar, J. Kopecký, and D. Fensel, “WSMO-Lite: Lightweight Descriptions of Services on the Web,” to be published in *Proc. 5th IEEE European Conf. Web Services (ECOWS 07)*, IEEE CS Press, 2007.
6. D. Booth et al., *Web Services Architecture*, World Wide Web Consortium (W3C) note, 11 Feb. 2004; www.w3.org/TR/ws-arch.

Advancement of Structured Information Standards) Semantic Execution Environment Technical Committee (www.oasis-open.org/committees/semantic-ex) specifies the middleware services’ functionality. The WSMX (Web Service Modeling Execution Environment, www.wsmx.org) and IRS-III (Internet Reasoning Service, <http://kmi.open.ac.uk/projects/irs>) are the reference implementations of OASIS SEE specifications.

The SEE comprises three sublayers of middleware services. The *vertical sublayer* defines a

framework that functions across the other two sublayers but that remains invisible to them:

- *Execution management* defines the control of middleware services’ distributed execution.
- *Security* defines secure communication—that is, authentication, authorization, confidentiality, data encryption, traceability, or nonrepudiation support for execution scenarios.

```

01  ...
02  concept Customer
03      hasConnection ofType (1 1) NetworkConnection
04      hasService ofType (0 1) Service
05      isValidCustomer ofType (1 1) boolean
06  concept Service
07      requiresBandwidth ofType (1 1) integer
08  concept NetworkConnection
09      providesBandwidth ofType (1 1) integer
10  concept VideoOnDemandService subConceptOf Service
11  concept IPTelephonyService subConceptOf Service
12  ...

```

Figure 2. A simplified ontology for a hypothetical telecommunications-services system.

The *broker sublayer* defines the functionality necessary for goal-based discovery and invocation:

- *Discovery* defines tasks for identifying and locating business services that can achieve a goal.
- *Adaptation* defines an adaptation within a particular integration process according to users' requirements (for example, service contracting and validation).
- *Orchestration* defines the execution of a composite business process together with the communication between a service requester and a service provider in that process.
- *Monitoring* defines the monitoring of the business services' execution. It gathers information on invoked services—for example, for determining quality of service or for identifying faults during execution.
- *Fault handling* defines how to handle faults occurring during the business services' execution.
- *Mediation* defines interoperability at the data and process levels.
- *Composition* defines the composition of services into an executable workflow.
- *Grounding* defines transformations from semantic descriptions of business services to nonsemantic descriptions, and vice versa.

The *base sublayer* defines functionality that isn't directly required in goal-based discovery and invocation but that the broker layer requires for successful operation:

- *Formal languages* define semantic lan-

guages used for semantic description of services, goals, and ontologies.

- *Reasoning* defines reasoning functionality over semantic descriptions.
- *Storage* defines a persistence mechanism for various elements (for example, repositories for services and ontologies).
- *Communication* defines the middleware's inbound and outbound communication.

The business-services layer

The SESA adopts descriptions for business services using the WSMO service model while building on the underlying technology for invocation and communication. Here, we show how to define business services using descriptions of *information*, *functional*, *behavioral*, and *non-functional* semantics and how to ground those descriptions to WSDL message types and operations. (Although the semantic descriptions are independent of the underlying technology, the grounding to WSDL leverages existing de jure and de facto SOA standards in the SESA context.)

Information semantics. A business service's core is the information semantics (information model) on which it operates. This semantics is the formal definition of the domain ontology that the service uses for description of functional and behavioral semantics.

In general, to model ontologies, you can use *concepts*, *attributes*, *relations*, and *axioms*. Concepts describe classes of objects and define the terminology of the domain of discourse. For example, in figure 2, the **Service** concept (line 6) stands for the class of all services that can be put in a subsumption relation by means of the **subConceptOf** construct (lines 10 and 11).

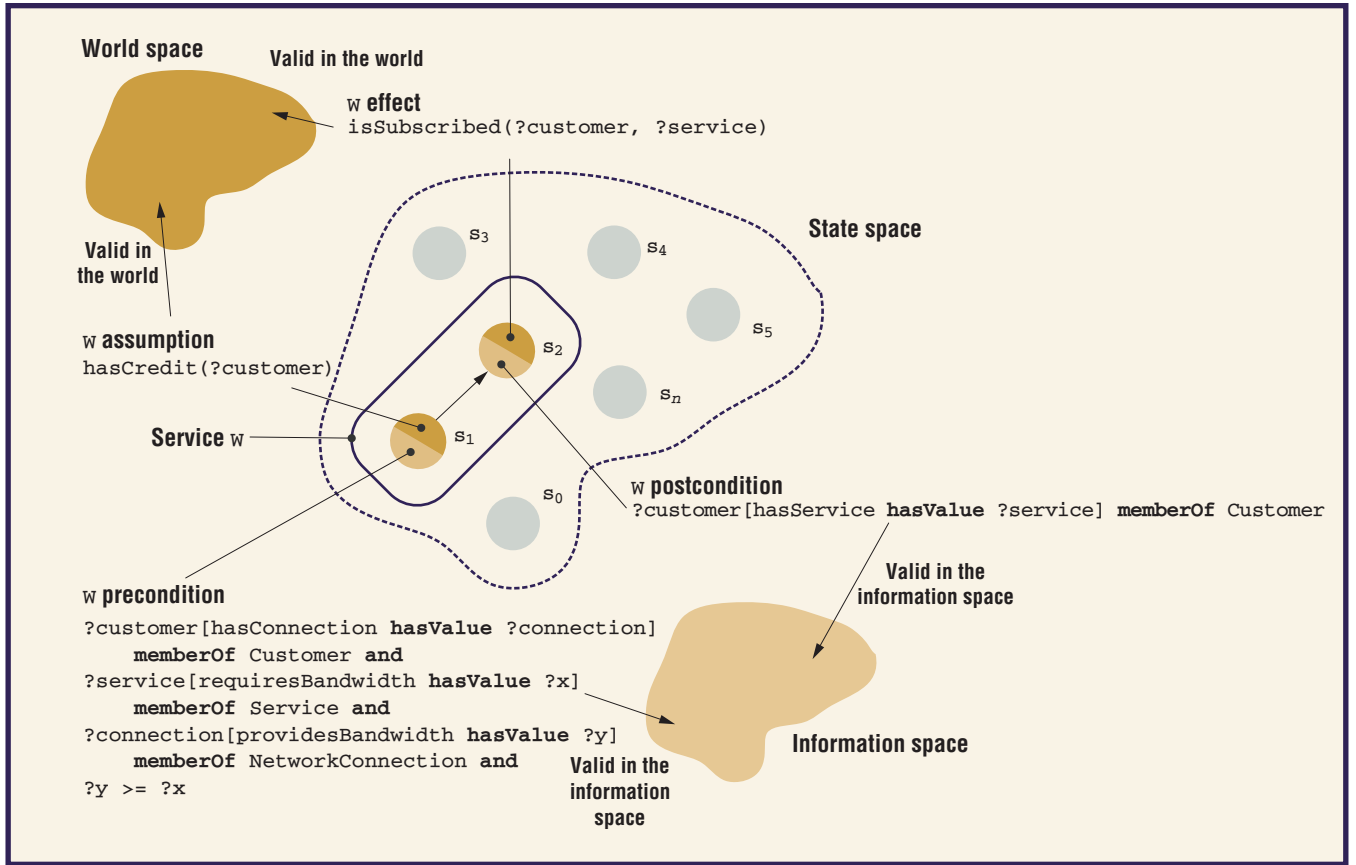


Figure 3. A functional-semantic example.

Attributes define relations between concepts, define these relations' constraints, and point to data types. For example, in figure 2, the *Customer* concept has the attributes *hasConnection* and *hasService*. These attributes point to concepts for the parts of the *Customer* concept with the restrictions that the network connection definition is mandatory while at most one connection can be defined.

Axioms define arbitrary complex logical expressions over other definitions in the ontology. We show examples of logical expressions for some descriptions of services later in this article.

Functional semantics. This semantics is the formal description of service functionality, describing what a service can offer to its users when it's invoked. We describe the service's functionality as a *capability*, which we model as *preconditions*, *assumptions*, *postconditions*, and *effects*. As figure 3 illustrates, preconditions and assumptions define the conditions of the information space and of the world outside that space, respectively, that must hold for state s_1 —the system's state before the invoca-

tion of the service W . Similarly, postconditions and effects define the conditions for state s_2 —the system's state after the invocation of the service W . We then define the invocation of W as a *transition* between states s_1 and s_2 :

$$\begin{aligned} &\text{holds}(\text{PRECONDITIONS} \wedge \\ &\quad \text{ASSUMPTIONS}, s_1) \\ &\rightarrow \text{holds}(\text{POSTCONDITIONS} \wedge \\ &\quad \text{EFFECTS}, s_2). \end{aligned}$$

In figure 3, the state space comprises all the services available for a search (for example, services in a repository). The information space comprises all the information available to all the services. The world space comprises the information outside the information space. Here, the capability uses WSMML to describe a network subscription service's functionality—in this case, video on demand (VoD). The precondition specifies that the customer must have a connection with the minimal bandwidth the service requires. The assumption specifies that the customer must have sufficient funds in his or her bank account. The postcondition identifies a valid customer hav-


```

01 ...
02 stateSignature
03   in
04     SearchCustomerRequest withGrounding
05       {"http://someuri/service.wsdl#wsdl.interfaceMessageReference(SCust/in0)"}
06     SubscriptionRequest withGrounding
07       {"http://someuri/service.wsdl#wsdl.interfaceMessageReference(AddS/in0)"}
08   out
09     Customer withGrounding
10       {"http://someuri/service.wsdl#wsdl.interfaceMessageReference(SCust/out0)"}
11     SubscriptionResponse withGrounding
12       {"http://someuri/service.wsdl#wsdl.interfaceMessageReference(AddS/out0)"}
13
14 transitionRules
15   forall {?searchCustomer} with (
16     ?searchCustomer memberOf SearchCustomerRequest )
17   do
18     add(_# memberOf Customer)
19   endforall
20
21   forall {?subscriptionRequest, ?customer} with (
22     ?subscriptionRequest[customerId hasValue ?cid] memberOf SubscriptionRequest and
23     ?customer[id hasValue ?cid] memberOf Customer )
24   do
25     add(_# memberOf SubscriptionResponse)
26   endforall
27 ...

```

ing the connection and service defined. The effect specifies that the customer subscribes to the service.

Behavioral semantics. This semantics defines a service's external and internal behavior. The external behavior, called a *choreography*, describes a protocol that a client uses to consume the service functionality. The choreography describes interactions involving messages sent to the service from the network and to the network from the service. (Our definition of choreography differs from the one that the Web Services Choreography Description Language² uses.) The internal behavior, called an *orchestration*, describes a workflow—that is, how the service's functionality is aggregated out of other services.

To model the choreography and orchestration, we use an *abstract state machine*. We define the ASM as

$$I_W = (In, Out, L),$$

where I_W is the interface of the choreography or orchestration, In and Out are ontological concepts of some information semantics used in input and output messages, and L is a set of rules. Each rule has the form “if *condition* then *effect*” and corresponds to one or more interactions with the service following the underlying *message exchange pattern* (we explain MEPs in more detail later). For the rule to execute, the condition must hold in a state in the information space; the effect defines how the state changes when the rule executes—that is, how the information space is modified by adding, deleting, or updating the data.

Figure 4 uses two rules to define the choreography for the service in figure 3. The first rule specifies how to get the details about the customer (lines 15–19). This rule can execute if SearchCustomerRequest is

Figure 4. An example choreography, which describes a service's external behavior.

Table 1

**Message exchange patterns, rules,
and Web Services Description Language operations**

MEP and rule	WSDL operation
in-out: if c_1 then $add(c_2)$ $c_1 \in \text{In}, \text{ref}(c_1, \text{msg1})$ $c_2 \in \text{Out}, \text{ref}(c_2, \text{msg2})$	<code><operation name="oper1" pattern="http://www.w3.org/ns/wsdl/in-out"></code> <code> <input messageLabel="In" element="msg1" /></code> <code> <output messageLabel="Out" element="msg2" /></code> <code></operation></code>
in-only: if c_3 then <i>no effect</i> $c_3 \in \text{In}, \text{ref}(c_3, \text{msg3})$	<code><operation name="oper2" pattern="http://www.w3.org/ns/wsdl/in-only"></code> <code> <input messageLabel="In" element="msg3" /></code> <code></operation></code>
out-only: if <i>true</i> then $add(c_4)$ $c_4 \in \text{Out}, \text{ref}(c_4, \text{msg4})$	<code><operation name="oper3" pattern="http://www.w3.org/ns/wsdl/out-only"></code> <code> <output messageLabel="Out" element="msg4" /></code> <code></operation></code>
out-in: if <i>true</i> then $add(c_5)$ if $c_5 \wedge c_6$ then <i>no effect</i> $c_5 \in \text{Out}, \text{ref}(c_5, \text{msg5})$ $c_6 \in \text{In}, \text{ref}(c_6, \text{msg6})$	<code><operation name="oper4" pattern="http://www.w3.org/ns/wsdl/out-in"></code> <code> <output messageLabel="Out" element="msg5" /></code> <code> <input messageLabel="In" element="msg6" /></code> <code></operation></code>

in the information space (in our example, we assume that the user will supply this information, as we explain later). The second rule depends on the first rule and defines how to perform the service subscription (lines 21–26). Here, the subscription request must use the customer’s identifier (`customerId`).

Nonfunctional semantics. This semantics describes additional constraints over service functionality that the functional description didn’t capture. They’re incidental details specific to a service’s implementation or running environment, independent of the service’s actual purpose but necessary for successful communication. To express these constraints, we can use policy languages (for example, WS-Policy³) and their semantic representation.⁴ Nonfunctional descriptions can also capture other information about services, such as the author or version, using the Dublin Core metadata set (<http://dublincore.org>). In addition, we use them for grounding, as we describe next.

Grounding. The SESA business services adopt the WSMO semantics for various descriptive parts of services. For the parts regarding service invocation (such as how and where the client can access the service), the WSMO defines grounding from the semantic descriptions to the WSDL descriptions. The SESA uses this grounding for on-the-wire message

serialization (WSDL binding), physical Web service access (WSDL service and end point), and communication (SOAP).

In addition, business service descriptions don’t preclude the use of other relevant Web services specifications (WS-*) such as policy, service security, reliable messaging, and transaction. While policy specifications can have semantic representations as nonfunctional descriptions, the other specifications can serve as part of the WSDL descriptions. To support such specifications at the SESA middleware layer, relevant middleware services should implement them.

The grounding specifies

- *references* for ontological concepts used in a choreography to WSDL messages and
- *transformations* from XML data to instances of ontological concepts (lifting) and vice versa (lowering).

The grounding is useful during communication with the service after the definition of WSDL operations and their MEPs.

Table 1 shows four basic types of WSDL 2.0 MEPs (www.w3.org/TR/wsdl20-adjuncts/#meps) and their corresponding choreography rules and WSDL operations. Here, c_1, \dots, c_6 are ontological concepts specified as input (In) or output (Out) of some choreography I_w . `msg1, ..., msg6` are XML Schema elements of input or output messages of operations.

```

01 ...
02 Goal GoalSubscription
03
04 capability
05   precondition
06     definedBy
07       ?customer[hasConnection hasValue ?connection] memberOf Customer
08   postcondition
09     definedBy
10       ?customer[hasService hasValue ?service] memberOf Customer and
11       ?service memberOf VideoOnDemandService
12
13 choreography GoalSubscriptionChoreography
14   stateSignature
15     in
16       OrderResponse withGrounding
17         {"http://someuri/goal.wsdl#wsdl.interfaceMessageReference(SS/in0)"}
18     out
19       OrderRequest withGrounding
20         {"http://someuri/goal.wsdl#wsdl.interfaceMessageReference(SS/out0)"}
21
22   transitionRules
23     forall {?request} {
24       naf ?request memberOf OrderRequest )
25     do
26       add(_# memberOf OrderRequest)
27     endForall
28
29     forall {?request, ?response} with (
30       ?request memberOf OrderRequest and
31       ?response memberOf OrderResponse )
32     do
33     endForall
34 ...

```

Figure 5. An example goal.

$ref(c, m)$ denotes a reference grounding of an ontological concept c and a message msg .

As figure 5 shows, more complex rules usually exist whose conditions define the order in which the invocation should happen. The choreography specification's current version doesn't handle other types of MEPs, such as ones including faults and optional messages.

Figure 4 shows the example for the reference grounding (lines 5, 7, 10, and 12). A lifting or lowering grounding defines a transformation (usually in XSLT, Extensible Stylesheet Language Transformations) that's specified as a URI (uniform resource identifier) in the lift-

ing or lowering nonfunctional description of the service (not shown in figure 4). When the rule for obtaining customer details (lines 15–19) executes (the condition holds in the information space), the system first lowers the instance data of `SearchCustomerRequest` to the XML message referenced by the concept (line 5). The system then invokes the corresponding WSDL operation, passing the XML message as the input data. Next, the system lifts the output XML message received from the invocation to the instance of the referenced `Customer` concept (line 10). Finally, the system adds the resulting data to

When no single service can satisfy the whole goal, the composition task tries to create a plan for that goal.

the information space according to the rule's action (line 18).

The problem-solving layer

Through this layer, users can formulate or identify goals, submit goals, interact with the architecture during processing, and get desired results. End users can perform these activities through some domain applications; engineers can perform them through some management tools—that is, an integrated development environment. The reference implementations of the IDE framework developed for the SESA are the Web Service Modeling Toolkit (WSMT) and WSMO studio (www.wsmostudio.org).⁵

The key to this layer is the formal description of the user's objectives. For this purpose, the SESA defines the goal description as a requested capability (what the user wants to achieve) and a choreography interface (how the user wants to communicate) having the same structural definition as the business service (see the sections "Functional semantics" and "Behavioral semantics"). In addition, we can express user requirements such as a service's desired quality by using nonfunctional descriptions of the goal.

Figure 5 shows a goal of a user who wants to subscribe to a VoD service. The precondition (lines 5–7) specifies that the user is a customer already having some network connection. The postcondition (lines 8–11) specifies that the customer subscribes to the intended VoD service. The choreography interface defines which data the user expects to supply (specified in `OrderRequest`) and to receive (specified in `OrderResponse`). The rules in lines 23–33 correspond to the out-in MEP of the underlying WSDL (see table 1). Here, `naf` (negation as failure) is a non-monotonic inference rule that derives `not OrderRequest` when `OrderRequest` fails to be derived in the information space.

Integration

The SEE middleware defines two phases for business services integration: *late binding* and *execution*.

Late binding

This phase allows binding a user goal and a set of business services by means of intelligence in the SEE middleware. We call this phase "late binding" because the binding isn't known a priori (that is, during modeling) and can be per-

formed in a semiautomated way on the fly. Typically, late binding involves tasks for service discovery, contracting, and validation, complemented by composition and interspersed with mediation. This phase operates on semantic descriptions of business services and can take several forms, depending on the scenario.

Discovery. When the system receives a user goal, the discovery task matches the goal capability with the capabilities of services in the middleware repository. Taking the goal from figure 5 as an example, this task finds a potential service satisfying the goal capability in figure 3 (the goal postcondition concept `VideoOnDemandService` subsumes the `Service` concept from the service postcondition). The discovery task must evaluate various semantic relationships between the goal and services, including exact match, intersection match, subsumption match, plug-in match, and disjointedness.⁶

Contracting. This process refines the discovery results by consulting dynamic instance data of the goal and the potential service. Including such data in static service descriptions is clearly infeasible—the system must obtain the data dynamically during the late binding.

In our example, the goal data is part of the service request as shown in figure 5 (through the rule in lines 23–27), while the system obtains the service data by executing appropriate rules from the service choreography in figure 4 (that is, it obtains the customer details using the rule in lines 15–19). In this example, we assume that through the order request in figure 5, the user supplies the customer identification together with the details for the subscription request needed by the service choreography in figure 4. If the goal and service choreographies use different ontologies, data mapping and mediation are necessary. The potential service can satisfy the goal if all data is fetched and the service precondition evaluates to true in the information space. In our example, the precondition holds if the customer's connection bandwidth is greater than the service's required connection bandwidth. More information about this process appears elsewhere.⁷

Composition. When no single service can satisfy the whole goal, the composition task tries to create a plan for that goal. Using AI methods such

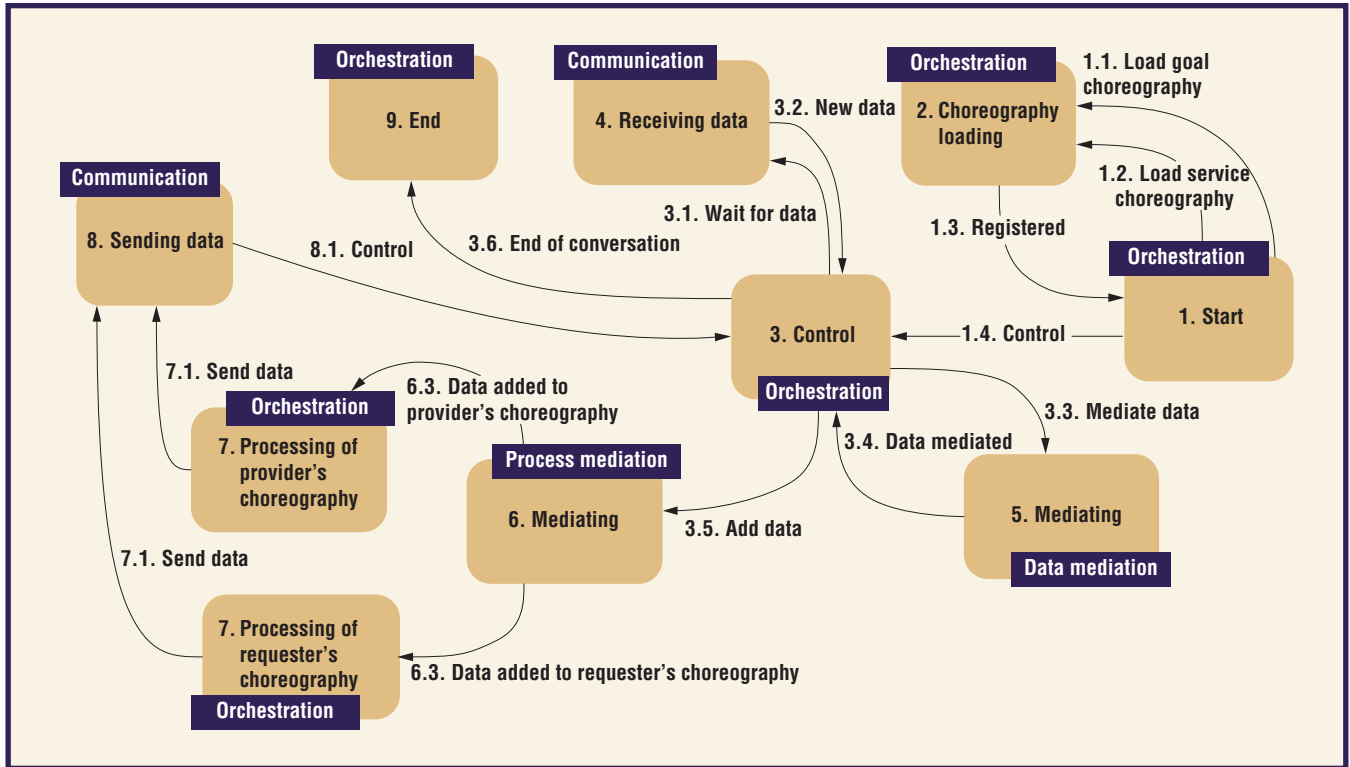


Figure 6. The control state diagram for the execution phase.

as backward or forward chaining, this task tries to find an order of services while recursively performing discovery and contracting. This results in a description of a composite business service's orchestration.

In our example, if the customer connection bandwidth is less than the service connection bandwidth, composition might result in a workflow of two services—that is, an upgrade connection service and the subscription service.

Validation. Although late binding aims to automate integration, human validation might be necessary in sensitive domains such as e-health, where late-binding results could have crucial real-world effects. In the SESA, a combination of service-analysis and user-approval methods ensures this validation.

Service analysis provides information based on the service usage and quality, answering questions such as “how often does the service fail?” and “how often is the service used in the given context?” With the help of this information, when the system notifies the user that the goal can't be fully satisfied, he or she can relax some of the functional or nonfunctional requirements. By selecting less functionality, the user might find higher-quality services; by re-

laxing the nonfunctional requirements, the user might find services with more functionality.

Execution

Figure 6 depicts this phase, which is performed by three middleware services. Orchestration manages the whole execution process, mediation resolves heterogeneity issues, and communication implements grounding together with inbound and outbound communication.

The process involves nine possible states:

1. The execution starts when the system receives a user goal (representing a requester) and a service (representing a provider). These descriptions are usually the result of the late binding. We also assume the software engineer has created the mapping of the goal and service ontologies and stored it with the middleware.
2. The system loads the goal and service choreographies into the orchestration engine, creating one instance for each. Each instance contains processing memory (representing the information space) used for storing data for runtime processing and a set of rules from the choreography definition.
3. The execution waits for new data (state 4),

About the Authors



Tomas Vitvar is a senior researcher at the Digital Enterprise Research Institute in Galway, Ireland. He received his PhD in computer science from the Czech Technical University. His research interests are in distributed systems and applications, including service-oriented computing, Semantic Web services, and enterprise computing. He's a member of the IEEE and of working groups in the World Wide Web Consortium and the Organization for the Advancement of Structured Information Standards. Contact him at the Digital Enterprise Research Inst., National Univ. of Ireland, IDA Business Park, Galway, Ireland; tomas@vitvar.com.

Michal Zaremba is a postdoctoral researcher at the Digital Enterprise Research Institute in Innsbruck, Austria, where he leads the Semantic Execution Environment cluster, driving research on the system architecture for Semantic Web services. His research interests include Semantic Web services, e-business, enterprise application integration, business-to-business integration, and business process management. He received his PhD in industrial engineering from the National University of Ireland. He's a contributor to the Web Service Modeling Ontology and Web Service Execution Environment (WSMX) working groups and the chair of the Semantic Execution Environment technical committee of the Organization for the Advancement of Structured Information Standards. Contact him at Viktor-Franz-Hess Strasse 5 top 2, 6020 Innsbruck, Austria; michal.zaremba@deri.at.



Matthew Moran is a PhD student at the Digital Enterprise Research Institute in Galway, Ireland. His research interests are in Semantic Web services, and he's a coauthor of the Organization for the Advancement of Structured Information Standards Semantic Execution Environment Working Group specifications. He received his honours degree in electronic engineering from the National University of Ireland, Galway. Contact him at 25 Friars Hill, Rahoon, Galway, Ireland; matthew.moran@deri.org.

Maciej Zaremba is a PhD researcher at the Digital Enterprise Research Institute in Galway, Ireland, working on Semantic Web services. His main research interests are Semantic Web services, business-to-business integration, and business process management. He received his MSc from the Wroclaw University of Technology. He's a member of the Web Service Execution Environment (WSMX) working group, a WSMX developer, and a member of the Semantic Execution Environment Technical Committee of the Organization for the Advancement of Structured Information Standards. Contact him at the Digital Enterprise Research Inst., National Univ. of Ireland, IDA Business Park, Galway, Ireland; maciej.zaremba@deri.org.



Dieter Fensel is the scientific director of the Digital Enterprise Research Institute in Innsbruck, Austria. His research interests are the Semantic Web, Semantic Web services, and semantically enabled service-oriented architectures. He obtained his PhD in economic science from the University of Karlsruhe. He's a member of the IEEE Intelligent Systems advisory board. Contact him at the Digital Enterprise Research Inst., Univ. of Innsbruck, Technikerstraße 21a, 6020 Innsbruck, Austria; dieter.fensel@deri.at.

processes the available data (states 5–8), or ends the execution (state 9).

4. The system checks whether new data from the user or service (both follow their choreographies) is available. If the system receives new data, it lifts the data using particular grounding definitions.
5. When new data is available, the system passes it to the data mediator. This mediator mediates the data from the user to the


service ontology or from the service to the user's ontology.

6. Control then passes to the process mediator, which decides where to put the new data: into the processing memory of the user's choreography, the processing memory of the service's choreography, or both. This mediator bases its decision on evaluation of the rules' heads for each choreography. In particular, the mediator evaluates whether the subsequent processing of the choreography can use this data.
7. The system processes the updated choreographies. That is, for a rule whose head satisfies the content of the choreography's processing memory, the system executes the body.
8. For each input concept in the rule's condition, the system sends the data to the service or user, according to the definition of the underlying operation. To do this, the system lowers the data to the corresponding XML message. The execution then goes back to the control state (state 3).
9. When no data is available for the processing, no rules remain to be processed in choreographies, and no data remains to be received from the user or the service, the execution reaches the end state.

While the SESA facilitates a novel style of integration of services by means of semantic service descriptions and AI methods, some people say that such an approach isn't realistic today. They argue that the complexity of semantic languages and integration techniques that depend on logical reasoning is a burden for service processing and high performance.

However, the logical reasoning can efficiently help resolve inconsistencies in service descriptions as well as maintain interoperability when these descriptions change. The more complex the services' descriptions are, the more difficult it is for a human to manually maintain the integration. The semantics that promote the automation is the key to such integration's flexibility and reliability.

To demonstrate the value of semantics for service descriptions as well as automation in service integration, we're working on the Semantic Web Services Challenge (www.sws-challenge.org). The SWS Challenge aims to establish a

common understanding, evaluation scheme, and testbed to compare and classify various approaches to services integration in terms of their abilities as well as their shortcomings in real-world settings. Although a world full of services doesn't exist yet, one-click integration will be desirable. The SESA and its related activities enable such a world as well as such integration. 

Discuss this article at <http://blog.vitvar.com/SESA>.

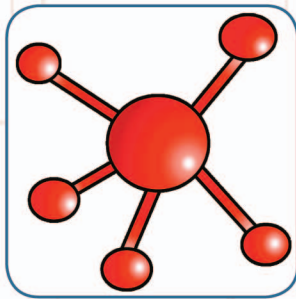
Acknowledgments

Science Foundation Ireland grant SFI/02/CE1/I131 and the EU projects Knowledge Web (FP6-507482), SemanticGov (FP6-027517), and SUPER (Semantics Utilised for Process Management within and between Enterprises, FP6-026850) supported this research. We thank Jacek Kopecky for his input on nonsemantic descriptions of services.

References

1. D. Roman et al., "Web Service Modeling Ontology," *Applied Ontology*, vol. 1, no. 1, 2005, pp. 77–106.
2. N. Kavantzaz et al., *Web Services Choreography Description Language Version 1.0*, World Wide Web Consortium (W3C) candidate recommendation, 9 Nov. 2005; www.w3.org/TR/ws-cdl-10.
3. A. Vadamuthu et al., *Web Services Policy Framework*, World Wide Web Consortium (W3C) recommendation, 4 Sept. 2007; www.w3.org/TR/ws-policy.
4. V. Kolovski et al., "Representing Web Service Policies in OWL-DL," *The Semantic Web—ISWC 2005*, LNCS 3729, Springer, 2005, pp. 461–475.
5. M. Kerrigan et al., "The Web Service Modeling Toolkit—An Integrated Development Environment for Semantic Web Services," *The Semantic Web: Research and Applications, 4th European Semantic Web Conf.*, LNCS 4519, Springer, 2007, pp. 789–798.
6. U. Keller et al., "Automatic Location of Services," *The Semantic Web: Research and Applications, 2nd European Semantic Web Conf.*, LNCS 3532, Springer, 2005, pp. 1–16.
7. T. Vitvar, M. Zaremba, and M. Moran, "Dynamic Discovery through Meta-Interactions with Service Providers," *The Semantic Web: Research and Applications, 4th European Semantic Web Conf.*, LNCS 4519, 2007, pp. 84–98.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.



IEEE DISTRIBUTED SYSTEMS ONLINE

IEEE Distributed Systems Online, the IEEE's first online-only publication, features free peer-reviewed articles as well as expert-moderated topic areas.

Topics include:

- Cluster Computing
- Grid Computing
- Web Systems
- Mobile & Pervasive Computing
- Middleware
- Security
- Parallel Processing
- Operating Systems
- Games & Simulation

<http://dsonline.computer.org>