

EFSOC: A Layered Framework for Developing Secure Interactions between Web-Services [†]

Willem-Jan van den Heuvel (wjheuvel@uvt.nl), Kees Leune (kees@uvt.nl) and Mike P. Papazoglou (mikep@uvt.nl)
Infolab, Tilburg University, The Netherlands

Abstract.

Enterprises are rapidly extending their relatively stable and internally-oriented business processes and applications with loosely-coupled enterprise software services in order to support highly dynamic, cross-organizational business processes. These services are no longer solely based on internal enterprise systems, but often implemented, deployed and executed by diverse, external service providers. The ability to dynamically configure cross-organizational business processes with a mixture of internal and external services, imposes new security requirements on existing security models.

In this paper, we address the problem of defining and enforcing access control rules for securing service invocations in the context of a business process. For this purpose, we amortize existing role-based access control models that allow for dynamic delegation and retraction of authorizations. Authorizations are assigned on an event-driven basis, implementing a push-based interaction protocol between services. This novel security model is entitled the Event-driven Framework for Service Oriented Computing (EFSOC). In addition, this article presents an experimental prototype that is explored using a realistic case study.

Keywords: business process, security, role based access control

1. Introduction and Rationale

Service-Oriented Computing (SOC) is the prominent paradigm for developing and deploying distributed enterprise applications using multiple fine-grained web-services that may be offered and executed by various providers.

This challenging way of building enterprise applications is effectively captured by an extension to the initial model for service oriented computing, and is called the extended Service Oriented Architecture (SOA)[25]. SOA adds the capabilities for service aggregation and service management to the architectural backbone of services-oriented computing.

Principally, SOA relies on universally accepted web-service standards to provide broad interoperability among different platforms, e.g.,

[†] This work has been partially funded by the Netherlands Organization for Scientific Research (NWO) as part of the PRONIR project.



SOAP (Simple Object Access Protocol) [6], WSDL (Web Service Description Language) [7] and UDDI (Universal Description, Discovery and Integration protocol) [29]. Additionally, SOA espouses the principle of loose coupling to separate the participants of distributed computing interactions, allowing modifications the interface of one participant in the exchange does not affect the other. The combination of these two core principles means that enterprises can implement web services without having virtually any knowledge of the consumers of those services, and vice versa.

The nature of the context in which service-oriented computing thrives imposes unique security requirements. Large scale interconnection of systems and services provided by several service providers, and run-time composition and invocation of services require more flexible security models than those that were introduced before.

To reduce security risks of interactions between web-services, which may be offered and deployed by diverse service providers, any security model must have strong security measures for ensuring that all communication between service provider and consumer is adequately safeguarded against unauthorized disclosure or manipulation. Hence, confidentiality and integrity of message contents and message protocols needs to be guaranteed. In addition to these basic security requirements, highly flexible access control mechanisms are of vital importance to ensure that while a complex business process flows from one activity to the next, only authorized actors can invoke supporting web-services.

The main contribution of this paper is a layered security framework facilitating definition and enactment of authorized invocation of service ports within the context of a particular business processes according to a set of security policy (access/control) rules, while building on top of 'lower-level', communication-level security standards as much as possible.

The remainder of this paper is structured as follows. In section 2, we rationalize the unique security requirements of the Service-Oriented Architecture and we analyze emerging security standards. In section 3, we then examine conventional approaches to access control and we identify strengths and weaknesses of those approaches in relation to a services environment. In the next section, we introduce a case study, which will be used to illustrate the framework.

The framework itself is developed in two steps: firstly, in section 5 the structure of the framework is outlined after which section 6 explains how the framework supports run-time execution of processes. Section 7 demonstrates a prototype implementation of the framework. Finally, section 8 presents the conclusions and points out some future research directions.

2. Security in the Web-Services World

Web service operations are characterized by the classic client/server model. This is a straightforward approach to distributed computing that provides the advantage that clients are coupled to the servers only via a contract mechanism. All providers must make their services available by publishing their contract and advertising their service. Since this contract is fully described by using WSDL [7], developers can construct clients using the contract information. The result is a Service Oriented Architecture (SOA) where all applications and services are viewed equally as service endpoints, regardless of their physical location or the protocol used to invoke them. The driving assumption behind service-oriented computing is that such a view lets developers treat applications as a network of services that can be chained together to create complex business processes more effectively and quickly.

Asynchronous message-based interactions are typical in an environment, like the SOA, where multiple highly distributed applications and services need to interact with each other, and which requires loosely coupled interfaces. Asynchronous messaging allows each communication operation between two processes to be a self-contained, stand-alone unit of work. Each intermediary in a multi-process communication can have its own distinct conversation with its sender and its recipient. To cater for this requirement, service-oriented computing defines an event-driven communications-infrastructure that transforms the Internet into an interactive network for making composite services and applications customer-available.

Since web-services should only grant access to sanctioned clients, security is a first order business consideration. A balanced approach to security is needed, taking into account not only security considerations at the level of the infrastructure, but also requirements that follow from business policies and processes.

There are several unique security-related characteristics that need to be addressed to develop secure business processes with web-services, including:

- Authorization capabilities are quickly emerging as a major security requirement for enterprises embarking on e-business. Clearly, simply knowing the requesters accessing service ports is not sufficient. Additionally, mechanisms are needed to authorize specific parties to access web-services in the context of a particular business processes.
- With authorization models, access to web-services is restricted to authorized requesters and applications in much the same way that

Web sites restrict access to authorized clients and applications. Authorization is required to determine various levels of access privileges of requesting service clients and applications.

- Authorization may require various levels of scope. Usually, authorizations are associated to specific operations of a web-service. However, there may be occasions when authorization for a web-service (or business process) as a whole is more appropriate, for example in case of informational services providing product catalogue information.
- In addition to the above, authorizations may be defined for roles played by subjects that are interacting in the course of a business process. In contrast with conventional models, authorizations may be *dynamically (re-)allocated* to subjects signifying the fact that they are allowed to produce or consume particular events. This advanced model of access control is schematically presented in figure 1. This figure depicts a business process comprising five web-services. Its internal control flow is coordinated using business events (the arrows in this figure), which may be sent and received by authorized roles (abstractions of subjects) only. In addition, ‘meta-’ events may be fired to modify security policies in business processes, e.g., allocating or retracting authorizations.

2.1. SOA STANDARDS FOR ENFORCING SECURITY FOR SERVICE-DRIVEN BUSINESS PROCESSES

The research that is developed herein builds on top of several existing SOA standards, including BPEL, SAML and XAMCL.

The Business Process Execution Language (BPEL) [2] is a platform-agnostic, block-structured workflow-like language for designing and running business processes that depend on a set of web services. BPEL assumes that all message exchange takes place in a point-to-point fashion. In other words, any business relation takes place between exactly two parties. The relationship between a service and a business process is described in terms of partner link types, which formalize the roles that each partner may play in a conversation. This approach allows to implement cross-business interactions as a large number of point-to-point message exchanges, without specifying a priori which role each business partner plays. A comprehensive overview of the main service composition languages, including a comparison, can be found in [26].

BPEL supports the exchange of messages between business partners, while treating security as an extension to the specification using several complementary standard languages, e.g., WS-Security [4]

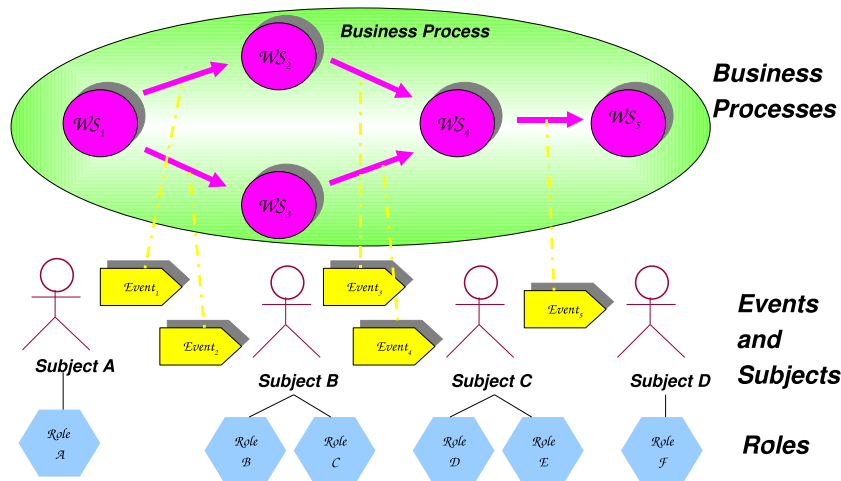


Figure 1. Securing Interactions between Web-Services via Authorizations on Events.

and WS-SecureConversation [9]. These standard languages help to secure message exchanges between parties just above the transport level (SSL), and are typically embedded in SOAP messages using encryption or signature headers.

SAML [11] entails another initiative that stems from the SOA domain. In contrast to WS-Security however, this language is not defined in conjunction with WSDL or BPEL. In a nutshell, SAML is a security assertion markup language which may be used to safeguard message exchange by allowing trading partners to share authentication and authorization information. SAML conveys assertions that may be validated by authorities. In addition to a set of predefined assertions, SAML offers a framework to define custom assertion types. Similar to WS-Security, SAML offers security at the level of message exchange, neglecting business semantics including the structure of business processes, events and roles.

XACML [14] has been developed independently from the SOA, aiming at the protection of enterprise data exchange. This standard has been ratified by OASIS, and essentially comprises a common XML security-policy language. This language enables administrators to specify access control rules to data and application resources. These rules can be defined as simple or more complex rules, using constructs such

as functions and data types. In addition, a decision language is provided to decide run-time whether or not access to a particular system resource may be granted.

Assessment of the above security standards in the context of the security requirements of SOA learns that they are mainly applied at the level of (XML) messages being exchanged between potentially authorized parties, regardless of rules and constraints that are defined in business processes, events and business rules. Moreover, these standards provide very little run-time support for querying and adapting security assertions, e.g., for load balancing, pricing or trust reasons; a critical requirement in SOA systems.

Unlike the above mentioned security standards, the EFSOC framework considers security considerations related to event-driven business processes, while abstracting from network and message level security issues. The latter security issues may be addressed by combining EFSOC with security standards including such as WS-Security in order to ensure message integrity and confidentiality. In addition, the framework offers a rich set of predefined queries to dynamically infer facts about security related issues, e.g., to retrieve which subjects are actually authorized to issue a particular event. These queries may be used as a the basis for dynamically making decisions about not only granting access to data resources, but also adapting service processes in a secure way. To our best knowledge, existing standards are not capable of such advanced inferencing capabilities.

3. Access Control Models

Access control entails a crucial design element for constructing secure business applications with web-services. Essentially, access control refers to the process of ensuring that resources are not used in an unauthorized way. Access control may be applied in SOAs for protecting against unauthorized invocations of operations of web-services in the context of a particular business process. Note however that access control as such does not address issues like message integrity and message privacy and authentication of service provider or service consumers. State-of-the-art security techniques based on public-key infrastructures, which are already in place and have been widely accepted, provide adequate safeguards for protecting the integrity and confidentiality of messages. Although these are important topics, they are not where the focus of this research is.

Two basic principles underly access control:

- *The Least Privilege Principle.* The least privilege principle states that a user should only have the absolute minimum of privileges to perform his job at that point in time [27]. Any additional privileges may be abused and can lead to circumvention of security mechanisms that are in place. The least privilege principle thus implies that the only interaction with services may take place in the context of an executing business process.

- *Separation of Duty.* This principle aims at preventing potential fraud by disseminating responsibilities for the execution of fragments of a business process among several participants. Separation of duty can be enforced both statically and dynamically. Static separation of duties is an approach that aims at avoiding conflicts that *may* introduce security risks if they are performed by the same subjects. Dynamic separation of duty may be enforced for avoiding conflicts which are caused by the execution of conflicting tasks. Similar to its static counterpart, four dimensions of conflicts need to be circumvented, viz. dynamically conflicting roles, dynamically conflicting duties, dynamically conflicting users and dynamically conflicting tasks. The interested reader is referred to [5] for an in-depth examination of structural and dynamic separation of duties in the context of workflow systems.

Over the past few years, a number of access control models were developed including discretionary access control, mandatory access control, and role-based access control. We will review these security models briefly and discuss their relevance in the domain of SOAs in the following.

3.1. DISCRETIONARY ACCESS CONTROL

The prevalent model for access control is the discretionary access control model (DAC). DAC is a means of restricting access to objects based on the identity of subjects and/or groups to which they belong [10]. Despite the fact that DAC is widely adopted, it is a well known fact that there exist several fundamental shortcomings which make it less suitable for large-scale use in dynamic, heterogeneous environments [21]. Firstly, discretionary access control adopts the assumption that the owner of an object controls access to it. The application of DAC in SOAs implies that the owner of a service (the service provider), controls the access to the services, cutting service clients out. Another restriction of DAC that makes it less suitable for SOAs, is that it can not deal with additional security requirements of clients.

Secondly, the DAC approach regulates access by an using access control matrix. The access control matrix denotes a matrix in which all subjects are cross-referenced and permissions are defined for each possible combination of (object-operation)-subject pairs. While this may be a viable solution for small scale business processes that require a limited number of web-services, it will quickly become impractical and too costly for more complex business processes involving a large number of services.

3.2. MANDATORY ACCESS CONTROL

Mandatory Access Control (MAC), also referred to as Multilevel Access Control, originates from military applications and was designed to regulate the flow of information. In [24], MAC is defined as a means of restricting access to objects based on a mapping between the sensitivity of the information that is contained in the objects, and, the formal authorization of subjects to access this information. This security model forbids creators of information to manage access to them. Instead, access is controlled by a central security administrator who designs a hierarchy of security levels and administers the assignment of security levels between objects and subjects. If a subject is operating at a security clearance which is at least as high as the security classification of the object, access is granted.

MAC is not particularly suited for usage in SOAs because it adopts rather restrictive hierarchical security levels, which can be hardly altered. Instead, a more flexible approach is required that allows for dynamic (re-)definition of security policies, e.g., at the level of service operations. Moreover, the assumption that access is granted and enforced by a central administration, is conflicting with the peer-to-peer nature of SOAs.

3.3. ROLE-BASED ACCESS CONTROL

Role-based access control models ([12], [13], [28]) are a contemporary control technique that introduce an indirection layer between users and permissions, which logically separates the role that users play in an organization or process, from the subjects. This is achieved by assigning permissions to roles, and, roles to subjects. Permissions are used to specify the capability of subjects to perform a particular operation, e.g., specifying an incident report or assessing incidents. In the RBAC approach, roles are typically organized in role hierarchies and constraints can be defined between roles.

An implicit assumption of the role-based approach is that the combination between roles and permissions remains relatively stable in time

[5]. However, the highly volatile environment in which web-services collaborate for implementing business processes, requires an environment in which roles and permissions *are decoupled* to cater for changes in the allocation of permissions to roles. By decoupling roles from permissions, the principle of of least privilege can be enforced more effectively, enabling a near real-time, active security implementation.

Another obstacle for applying RBAC in SOAs is that, similar to MAC, RBAC prescribes that the security administration takes place in a single, centralized, location. At the same time, this security model assumes that access control rules are typically evaluated by a central security agency, which introduces additional limitations on their flexibility.

In summary, existing access control models suffer from serious shortcomings may severely hinder their applicability to the SOA domain, including the need for a central authority that is responsible for allocating authorizations, hierarchical (instead of networked) security models, and fixed permission-to-role assignments. The EFSOC framework that is introduced later in this article, tries to dissolve these problems allowing for dynamic (re-)definition and deduction of authorizations on a peer-to-peer basis. Before outlining this framework, we wish to introduce a running example, to demonstrate the workings of EFSOC.

4. A Case Study: Incidence Response Processes

Incidence response processes are critical for managing security threats to enterprise applications and repository systems in organizations. Whilst the number of threats has taken a giant leap, largely caused by an explosive number of viruses attacking computer systems over the Internet, national and international laws are emerging to regulate privacy and integration of (shared) information. Consequently, these processes need to be constantly monitored and adapted, possibly replacing or reconfiguring its services or security rules.

In order to prevent and respond to computer security incidents in a coordinated manner, while respecting and possibly anticipating new regulations and (inter)national laws, Tilburg University's (UvT) management team has devised a new cross-departmental team of security experts. This Computer Emergency Response Team is named UvT-CERT. To perform its tasks, the team has direct access to a large collection of information sources and has far-reaching authority to intervene in daily management of computer systems and networks.

Web-services and processes were developed for the UvT-CERT for assisting the team in incident reporting, incident analysis and incident responses. Figure 2 depicts the process that was devised for handling security threats. It is capable of dealing with threats in two complementary manners: pro-actively and reactively. Pro-active incident management receives input from the threat gathering activity (see top left in figure 2), whereas reactive management extracts information from reports that were filed by internal users (see top right in figure 2).

The security basic incident handling process includes the following activities:

- *Gathering Critical Information.* The security watcher monitors information bulletins that are regularly issued by large vendors or by other international security organizations. In addition to bulletins, a number of sensors deployed around various areas of the network may trigger information to be sent to the control application, including web logs and intrusion detection system alerts.
- *Recording Incident Reports.* Incident reports may be filed by help desks, based on an incoming request from a customer via various means, including email, fax or telephone. Information that should be captured includes the date and time of report, activity and discovery, systems affected, description of the problem and assigned staff.
- *Triage.* Using information distilled from the previous two processes, the security analyst assesses and evaluates the current situation in order to prioritize the incident and associate a threat level.
- *Coordinating Responses.* Depending on the priority and type of the incident, standard precautions may be taken. Specialized platform or system administrators are typically contacted during this activity, depending on the impact of the incident and given the required expertise to resolve the problem. Finally, mitigation or response strategies are to be developed under suspicion of the incident handler.
- *Incident Recovery.* During this activity, the mitigation or response strategy that was developed during the previous activity, is operationalized. Three basic types of strategies are considered: *Request Information*, *Disconnect User* and *Publish Bulletin*. The first strategy involves gathering more information about the incident,

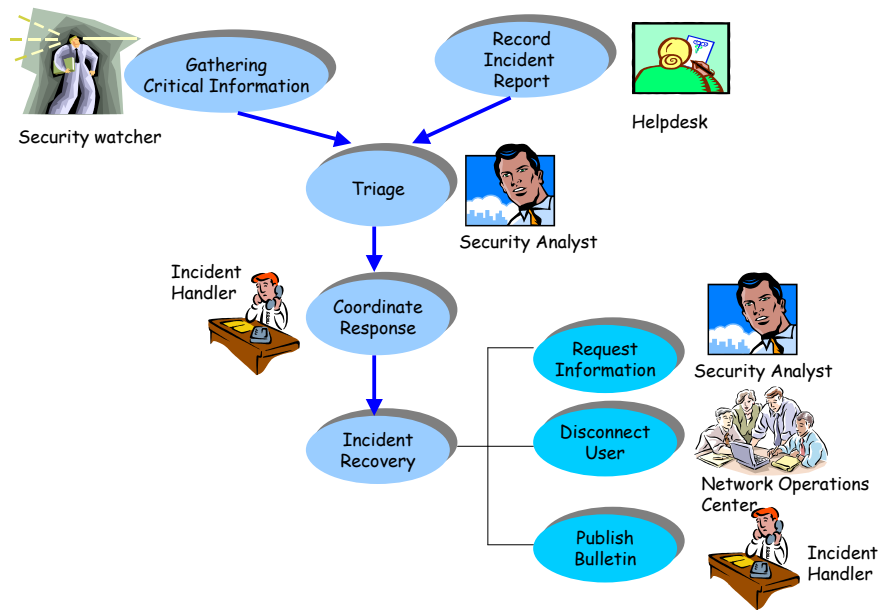


Figure 2. Basic Incident Handling Process.

possibly including forensic data gathering. Secondly, users may be disconnected from the network during which application or data may be restored at the affected site. In extreme cases, machines may be confiscated or physically isolated from the network. Lastly, incident handlers may pro-actively publish urgent security related news on internal message boards.

Each of the above activities is performed by a particular role, requiring one or more web-services. For example, the *Security Analyst* role is supported by a *Triage* web-service, that helps to create a comprehensive overview of the status of all university systems after a virus alert. Likewise, other roles are leveraged by one or more web-services.

The incidence response process is volatile in nature: subjects in the team, web-services used in the process, and process itself are constantly prone to minor modifications and more radical redesigns. For example, roles may be reallocated to new subjects, and new authorizations may be assigned to an existing or new role.

5. The EFSOC Framework

The Event-driven Framework for Service Oriented Computing, in short EFSOC, comprises a layered framework to develop business processes

with web-services. This framework takes into account authorizations and roles that web-services may play during the course of a business process.

In this section, we will first explain a meta-model, which formally defines the layers and their constituents. The layers of this model will be consecutively elaborated, illustrated and explored using the incident response case study.

5.1. THE UNDERPINNINGS OF THE EFSOC FRAMEWORK

The architectural layers in the EFSOC framework allow a logical separation of concerns. Separation of concerns reduces complexity by allowing each layer to define a particular aspect of the business process, e.g., one layer defines its activities whereas another layer defines access/control rules. In EFSOC, logical separation of concerns separates events from access right capabilities that constrain the usage of events by subjects. In addition, the EFSOC framework isolates the definition and enactment of services and processes from associated (typed) events. This facilitates not only specification and execution of events, processes and services, but also that of inferencing rules to enable run-time decision making processes revolving around new or adapted security policies.

Accordingly, the EFSOC framework encompasses three layers, each of which addresses specific business concerns. Figure 3 illustrates a UML metamodel of each layer, its key constituents and their interrelationships.

The *Event Layer* (see top left in this figure) provides constructs for defining events, and correlating events (such as an incoming alert) with basic event operation types, including send, reply, publish and subscribe.

The *Security Layer* builds on the event layer to enable a dynamic access control mechanism which can be used to constrain the ability of subjects to send or receive events. This layer is populated by three key building blocks: subjects, roles and access control rules. Subjects refer to actors who are skilled in performing a specific capability, e.g., evaluating an incoming security alerts. While participating in a business process, a subject can engage one or more roles. Each role refers to authorizations to invoke or execute specific service operations. Authorizations (permissions) are encapsulated in access control rules, that may in fact be reused across various business processes.

Finally, the *Business Process Layer* provides support for weaving web-services into business processes. In its most basic form, it constitutes two components: business processes and services. A service

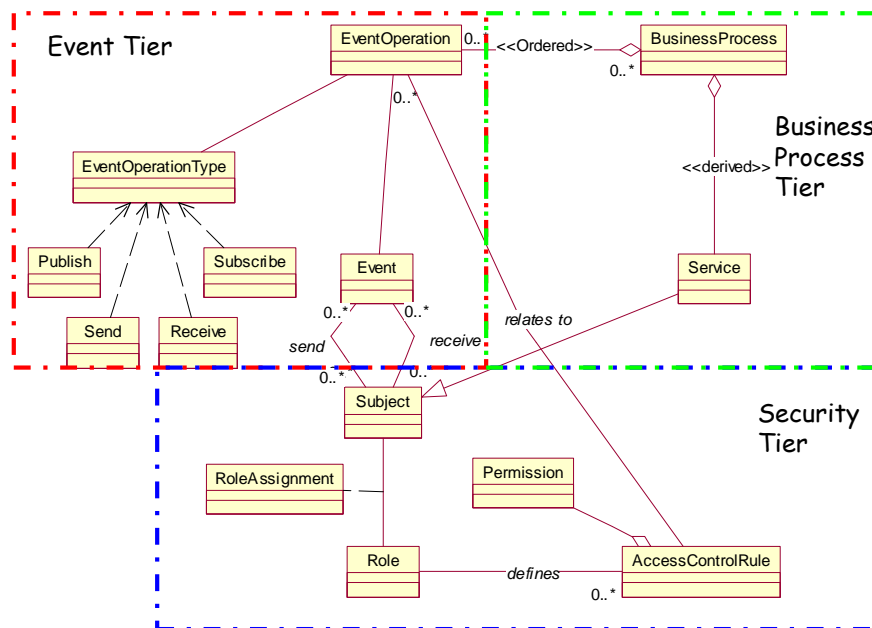


Figure 3. Meta-model of EFSOC.

defines operations that may be invoked by the process to which it is associated. Services encompass one or more service operations; for reasons of readability we have not included service operations in Figure 3.

As depicted in figure 3, services (and service operations) are not directly linked to a business process. This indirect relationship is expressed by the stereotype `<<derived>>` on the association between service and business process. This indirect relationship may be derived by associating business processes with event operations, event operations with events, events with subjects, and lastly, subjects with service operations in order to assure flexibility at the level of the access control rules. As we will demonstrate in section 6.2, this relationship may be determined in a straightforward manner by querying the metamodel.

In the remainder of this section, we will provide a more detailed description of the constructs of this metamodel.

5.2. THE EVENT LAYER

Events play a pivotal role in the EFSOC framework as they serve as the main fabric to which authorizations are dynamically assigned and checked. Events can be perceived as occurrences that happen over time, independently from each other. An event thus constitutes a “happen-

ing” that causes, or is caused by, an ‘impact’ on a business process [22]. As such, an event does not have a form or a shape. The ‘footprint’ of events may carry information, e.g., an incident report event can be parameterized with a date, time, request type, client.

In line with the findings in [20], we distinguish three aspects of an event. The first aspect constitutes the *form* of an event. An event is typically associated with data structure which describes information about the context in which the event occurred. The second aspect is the *significance* of an event. An event signifies an activity in the real world that has some minor or more profound implication or consequence. In EFSOC, this is materialized by assigning types to events. The third aspect of an event is its *relativity*, which represents the relationship in time or causality of one event to the other. In EFSOC, this is represented by the fact that an event is annotated with an ordered list of historic events.

In service-driven processes, events are typically materialized as messages that are produced or consumed by web services. Messages encompass two main elements: an *envelope* and a *body*. The envelope contains meta-information for routing the message (event) and evaluating authentication rules. The relativity and the significance of an event are expressed in the envelope by defining corresponding message headers. The type of the event is modelled by the event body, which incorporates a data structure representing the context in which the event was generated.

Traditional point-to-point connections between web-services are no longer the preferred method of communicating events, as it incurs an exponential increase in complexity when new web-services are added to a business process. Therefore, the EFSOC framework assumes that interactions between web-services are not limited to point-to-point interactions, allowing one-to-many, many-to-one and many-to-many connections, possibly in an a-synchronous mode.

These types of communication may be implemented according to the publish/subscribe protocol. The publish/subscribe protocol places a broker between sending and receiving services to manage subscriptions, and additionally to take care of routing and transformation of events. By doing so, this model gracefully allows asynchronous communication between sending and receiving services without specifying the recipients, making it a perfect candidate for developing flexible service-driven processes.

EFSOC distinguishes four basic “meta-” operations on events, named *event operation types* in figure 3, in order to realize the publish/subscribe protocol. The following four elementary event types are discerned: *publish*, *subscribe*, *send*, and, *receive*. The publish event operation type

```

<efsoc:eventoperationtype id="eot1">
  <efsoc:name>publish</efsoc:name>
</efsoc:eventoperationtype>

<efsoc:eventoperationtype id="eot2">
  <efsoc:name>subscribe</efsoc:name>
</efsoc:eventoperationtype>

<efsoc:eventoperationtype id="eot3">
  <efsoc:name>send</efsoc:name>
</efsoc:eventoperationtype>

<efsoc:eventoperationtype id="eot4">
  <efsoc:name>receive</efsoc:name>
</efsoc:eventoperationtype>

```

Figure 4. EDL specification defining the basic event operations in EFSOC

may be used to advertise services to a security service. Subjects may subscribe to published services using the subscribe event operation type. Once subscribed, clients will automatically receive messages that were sent by the publisher through the security service. The security service entails a specific category of middleware for facilitating secure interactions between services within the context of EFSOC. We will describe more details of this infrastructure service in section 7.

5.2.1. *The Running Example: Events, Event Operations and Event Operation Types*

To facilitate the specification of model elements and queries, a declarative XML syntax called EDL (EFSOC Definition Language), has been specified using the XML Schema notation [15].

Figure 4 shows an EDL excerpt that represents the definition of the four elementary events operation types in EFSOC. The complete XML schema definition of EDL is available in appendix A.

We now exemplify the definition of events according to the publish/subscribe protocol in EDL (see figure 5). For this purpose, we will focus on the *Disconnect User* activity in the basic incident handling process (see section-4). This activity is executed by the Network Operations Center in urgent situations, e.g., in case the security analyst has diagnosed the system and found a Trojan horse virus. The activity commences after receipt of a **Block** event, conveying the **ip_address** that needs to be blocked, and the **start_date** and **end_date** of closing off the Internet address. In addition to this information, the **Block** event

```

<!-- block event -->

<efsoc:eventbody id="eb1">
  <xsd:element type="xsd:string" name="ip_address"/>
  <xsd:element type="xsd:date" name="start_date"/>
  <xsd:element type="xsd:date" name="end_date"/>
  <xsd:element type="xsd:string" name="incident_id"/>
</efsoc:eventbody>

<!-- publish block event -->

<efsoc:eventoperation id="eo1">
  <efsoc:eventid>eb1</efsoc:eventid>
  <efsoc:eventoperationtype>eot1</efsoc:eventoperationtype>
</efsoc:eventoperation>

<!-- subscribe block event -->

<efsoc:eventoperation id="eo2">
  <efsoc:eventid>eb1</efsoc:eventid>
  <efsoc:eventoperationtype>eot2</efsoc:eventoperationtype>
</efsoc:eventoperation>

```

Figure 5. An EDL representation of a Blocking Event

also transports the `incident_id` that was allocated to a particular incident by the security analyst.

Figure 5 basically consists of three parts. Firstly, the event body of the event type `Block` is specified. This event body is named `eb1` and encapsulates four typed elements according to the above scenario. The event envelope is omitted from the XML description for purposes of brevity. Next, the `Block` event is published to the security service, after which one or more subjects may `subscribe` to this event type.

5.3. THE SECURITY LAYER

SOA perceives business processes as collections of web-services that collectively implement a business objective [25]. A business process thus reveals how individual operations of web-services are imported and linked to another. Such service invocations take place in a highly dynamic and rapidly changing environment, in which service requesters and providers are unknown to each other and (new) services may be included, updated or replaced in an ad-hoc manner.

Unlike conventional web-service standards, EFSOC does not primarily address the message transport layer, e.g., ensuring message confidentiality and integrity, but aims at ensuring that authorizations to invoke web-services may be determined dynamically, depending on the state of a business process (see section 3). For example, in case the basic incident handling process takes place in the **Triage** state, the Network Operations Center may not (yet) be allowed to invoke a service to **Disconnect a User**.

The security layer expresses access control rules that exactly prescribe the conditions under which subjects may send or receive messages, for invoking or executing a service. This layer serves as the foundation to dynamically infer whether or not subjects are allowed to send, receive, publish or subscribe to specific events.

The security model of the EFSOC framework assumes that subjects are assigned to roles. In contrast to RBAC, however, roles are not hardwired with a set of authorizations to perform functions. Instead, EFSOC introduces an additional mapping between authorizations (permissions) and roles. In this way, roles may be dynamically redefined and reassigned to meet new or changed security requirements, e.g., authorizations can be dynamically delegated or retracted. Hence, whilst classic RBAC models define $\langle \textit{subject}, (\textit{role} - \textit{authorization}) \rangle$ tuples, the EFSOC framework adopts $\langle \textit{subject}, \textit{role}, \textit{authorizations} \rangle$ triples.

This additional level of indirection between permissions and subjects who wish to perform a certain operation, is leveraged by the introduction of events and authorization rules. It is motivated by the assumption that roles and role-permission assignments are not stable in SOAs, but constantly in flux.

One of the biggest challenges of implementing security in a service-oriented environment is the fact that services are typically not only identified and combined, but also replaced in an unplanned, and possibly ad-hoc manner. Consequently, it may not be possible to predict in advance which service providers are active in the context of a particular process instance, nor which authorizations they should have.

EFSOC addresses the potential unawareness about providers of services by introducing three levels of authentication. Subjects may be unauthenticated, which implies that they are completely unknown. *Unauthenticated* subjects are typically restricted to accessing publicly available information. When a subject offers some form of valid and correct credentials to the EFSOC framework, the subject is considered to be partially authenticated. Finally, when a *partially authenticated* users activate one or more roles, he is considered to be *fully autho-*

rized. Depending on the level of authorization, subjects can be granted different permissions.

5.3.1. *The Running Example: Defining Roles, Subjects and Access Control rules*

The running example identifies the following five roles: **Security Watcher**, **Security Analyst**, **Helpdesk**, **Incident Handler**, and, **Network Operations Center (NOC)**. In addition, the basic incident handling process in the running example is staffed by the following two subjects: **John** and **Jane**. As highlighted in the above, the EFSOC framework allows us to allocate one or more roles to each subject. We have defined several roles, subjects and role assignments in figure 6. For example, the first `<roleassignment>` expresses that **John** participates in the `NOCRole`, and the second that **Jane** plays the `IncidentHandlerRole`.

Next, we specify an access control policy capturing two access control rules. Firstly, the access control rule `acr1` in figure 7 allows the role `r2` (assigned to **Jane**) to invoke the event operation `eo3` (hence, to issue a `Block` message). The second access control rule (`acr2`) articulates that role `r1` (played by **John**) is permitted to receive the `Block` message.

5.4. THE BUSINESS PROCESS LAYER

Business processes involve long-running interactions between trading partners to receive, invoke and reply to business activities. As alluded to before, EFSOC perceives business processes as event-driven entities. Hence, business processes are not interpreted as logical sequences of business activities, but as an ordered set of `EventOperations`. In the EFSOC framework, business processes are leveraged by a collection of subjects that are generating and consuming a logically ordered set of events, using event operations. As such, business processes provide the desired flow of event operations, while access control rules limit the ability to execute them.

5.4.1. *The Running Example: Specifying the Disconnect User Process*

The EFSOC framework is agnostic with regard to business process specification languages, such as BPEL or WSCI [3]. Actually, any of them may be applied to specify processes as long as they are capable of wiring events into process layouts. For reasons of simplicity, we herein abstract from specific internal details of the `Disconnect User` process (see the EDL excerpt in figure 8). We simply treat this process as a labelled collection, `bp1`, embracing two event operations: `eo3` for sending a block event, and, `eo4` for receiving a block event.

```

<efsoc:role id="r1">
  <efsoc:>NOCRole</efsoc:rolename>
</efsoc:role>

<efsoc:role id="r2">
  <efsoc:rolename>IncidentHandlerRole</efsoc:rolename>
</efsoc:role>

<efsoc:subject id="s1">
  <efsoc:subjectname>john</efsoc:subjectname>
</efsoc:subject>

<efsoc:subject id="s2">
  <efsoc:subjectname>jane</efsoc:subjectname>
</efsoc:subject>

<efsoc:roleassignment id="ra1">
  <efsoc:subjectid>s1</efsoc:subjectid>
  <efsoc:roleid>r1</efsoc:roleid>
</efsoc:roleassignment>

<efsoc:roleassignment id="ra2">
  <efsoc:subjectid>s2</efsoc:subjectid>
  <efsoc:roleid>r2</efsoc:roleid>
</efsoc:roleassignment>

```

Figure 6. EDL definitions of roles, subjects and role assignments.

6. Dynamic behavior of the EFSOC framework

In section 5, the three layers in which the static components of the EFSOC framework are defined were introduced. These layer serve as the fundament on top of which secure business processes may be executed.

In this section we will reveal the runtime, dynamic behavior to the framework. Section 6.1 introduces a service taxonomy in which a distinction is made between infrastructure services and operational services. Infrastructure services are services which implement the structural elements of the EFSOC model. Operational services are services implement business-driven functionality, depending on the infrastructures services for safeguarding secure (allowed) communications and interactions. In section 6.2, we then outline the way in which the specifications in the EFSOC framework may be queried to facilitate run-

```

<!-- allow CSIRTRole to send block event -->

<efsoc:accesscontrolrule id="acr1">
  <efsoc:permissionid>allow</efsoc:permissionid>
  <efsoc:eventoperationid>eo3</efsoc:eventoperationid>
  <efsoc:roleid>r2</efsoc:roleid>
</efsoc:accesscontrolrule>

<!-- allow NOCRole to receive block event -->

<efsoc:accesscontrolrule id="acr2">
  <efsoc:permissionid>allow</efsoc:permissionid>
  <efsoc:eventoperationid>eo4</efsoc:eventoperationid>
  <efsoc:roleid>r1</efsoc:roleid>
</efsoc:accesscontrolrule>

<!-- access control policy for block event -->

<efsoc:accesscontrolpolicy>
  <efsoc:eventid>eb1</efsoc:eventid>
  <efsoc:accesscontrolruleid>acr1</efsoc:accesscontrolruleid>
  <efsoc:accesscontrolruleid>acr2</efsoc:accesscontrolruleid>
</efsoc:accesscontrolpolicy>

```

Figure 7. EDL definitions of Access Control Rules.

```

<!-- business process -->

<efsoc:businessprocess id="bp1">
  <efsoc:eventoperationid>eo3</efsoc:eventoperationid>
  <efsoc:eventoperationid>eo4</efsoc:eventoperationid>
</efsoc:businessprocess>

```

Figure 8. EDL definition of the Disconnect User Process.

time execution of business processes, possibly dispersed over multiple, distributed locations.

6.1. THE EFSOC SERVICE TAXONOMY

Figure 9 introduces a services taxonomy on the level of the framework. In [19], two separate categories of services are distinguished:

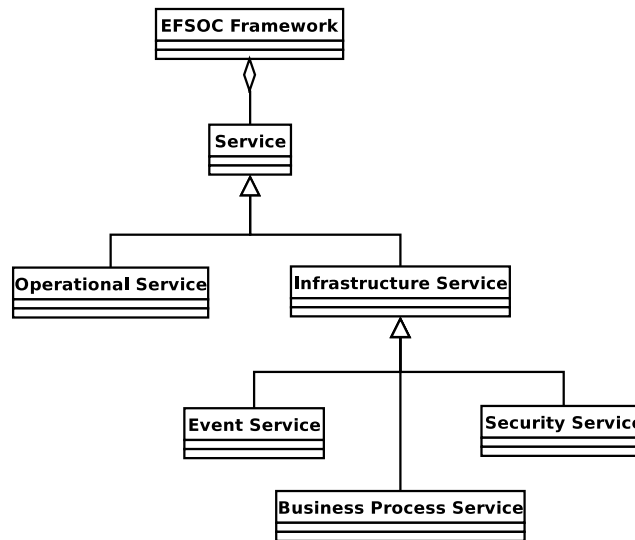


Figure 9. The EFSOC service taxonomy

- *Infrastructure services* enact the structural constituents of the EFSOC framework. Three separate types of infrastructure services are identified: a *security service*, an *event service* and a *business process service*. The security service, provides facilities for specifying access control policies, access control rules and permissions and provides the runtime enforcement of these rules. The event service allows management of subjects, roles and role assignments and provides all event-relay facilities, such as store-and-forward of events, and event queuing. Finally, the business process service acts as an enactment engine and monitor ensuring that business processes flow through service invocations in a controlled fashion.

As mentioned earlier in this paper, EFSOC is policy-neutral, which means that the framework does not impose a way of working. Instead, the EFSOC framework can be extended and configured to fit in many different organizational structures.

- *Operational services* provide the business implementations of service, and run on top of infrastructure services. Operational services are typically defined when business processes need them to provide certain functionality. The operational services which exists in the scope of the running example will be discussed later in this section.

Each of the three infrastructure services has been implemented as a SOAP-driven web service. Combined with a common programming interface, these web services provide the basic infrastructure for imple-

menting solutions according to the EFSOC framework. WSDL definitions of the service interfaces may be found in [18].

6.2. QUERYING THE FRAMEWORK

Once business processes and authorization rules are generated, queries may be constructed or reused to dynamically deduce information regarding the authorization of subjects to send, receive, publish or subscribe (to) particular events. These queries can be conceptualized by traversing the association between EFSOC types in the metamodel (see figure 3).

We have used First order predicate logic to infer new facts from the meta-model. We will demonstrate the overall working of *authorization-based* queries borrowing some definitions from the running example. Let us assume that the objective is to define a query which determines which access control rules apply to a given subject. To achieve this, the following logical propositions are defined according to the core EFSOC constructs:

- $\text{subject}(ra, s)$, denotes that s is the subject in a role assignment ra :
- $\text{role}(ra, r)$ designates that r is the role in role assignment ra ;
- $\text{rule}(ru, r)$ indicates that r is a role which is referenced from access control rule ru .

After the logical assertions have been postulated, logical inferencing rules can be defined “querying” these assertions and returning access/control rules that are assigned to a particular subject. This query may be formulated as follows:

$$\begin{aligned} \forall ru, s \quad & \text{subject}(s) \wedge \text{accesscontrolrule}(ru) \wedge \\ & (\exists r, ra \quad \text{roleassignment}(ra) \wedge \text{role}(r) \wedge \text{subject}(ra, s) \wedge \\ & \text{role}(ra, r) \wedge \text{rule}(ru, r)) \\ \implies & \text{rulesApplyingToSubject}(ru, s) \end{aligned}$$

The rule first determines which roles subject s is allowed to play by checking which role assignments are known for this subject. Thereafter, roles may be mapped to subjects. In the next step, it is inferred which access control rules are connected to these roles. Once, the mappings between rules and roles have been established, it can be easily concluded which control rules are attached to a particular subject.

In addition to this rather basic rule, a number of more sophisticated logical inferencing rules have been developed. These include queries to

determine which services play a role in a particular business process and queries which are used for access control purposes. Once new instances are added dynamically, or when changes to existing instances are made, the queries can be reused to (re-)check whether the security rules are satisfied by the entities populating the EFSOC framework.

The logical query language has been wrapped in XML using the XML Common Logic format (XCL) [1], and included in the EDL language. XCL is a XML markup language designed as a concrete (serialization) syntax for Common Logic [8] that is a first-order logical language. The structure of a query in EDL is formalized using XML Schema in appendix A. The EDL counterpart of the above query is verbose and space-consuming, so we have included it in appendix B.

Just like its predicate logic counterpart, the EDL query is designed to retrieve all access control rules that are associated with a particular subject, with name `subjname`. This query incorporates a unique identifier (`rulesApplyingToSubjectQuery`) and allows zero or more typed input parameters (here thus the name of the subject), followed by a formal rule which defines the actual query.

7. A Prototypical Toolset for Supporting EFSOC

The EFSOC framework, including its supporting EDL language, was partially validated using an experimental prototype. The main architecture of the experimental prototype is shown in figure 10. As can be seen in this figure, the core of the prototype implementation constitutes three web-services making up the security infrastructure service, the business process infrastructure service and the event infrastructure service. These services collectively implement the EFSOC framework on top of which other applications (e.g., an application supporting the basic incident handling process) may be developed.

The web-services are implemented in Java and may be either deployed on a Apache Tomcat server, or on a Java application which acts as a GUI for defining new roles, subjects and role assignments. The WSDL definitions of these services were published in [17]. Subject credentials and descriptive information are stored in a central directory which can be queried via the Lightweight Directory Access Protocol (LDAP) [30]. Finally, we developed a rule repository, which is stored in the filesystem of the server.

Due to its meta-modeling and inferring capabilities,

We have used ConceptBase [16] as the platform of choice to model the core elements of the EFSOC framework and to act as a data repository for events and role assignment. This decision was primarily based

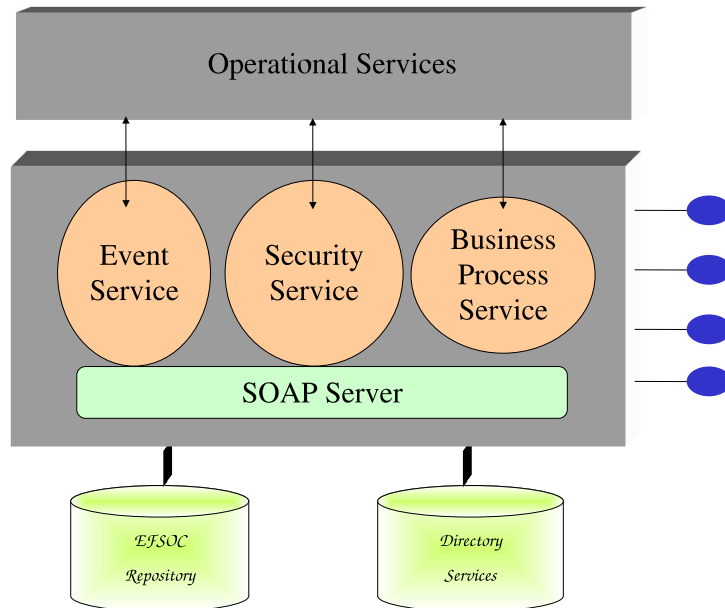


Figure 10. Architecture of the Experimental Prototype.

on the advanced meta-modeling and inferencing capabilities of this deductive database repository system. The underlying representation language of ConceptBase is O-Telos. O-Telos is an extension of the Telos language, that was designed at the beginning of the 90s [23] for supporting software development during all phases of the lifecycle, from analysis down to implementation and maintenance. Telos is a formal knowledge representation language, capable of storing (temporal) facts, e.g. about analysis models, and deductively inferring new facts. This language is based on first order logic and has a frame syntax to represent and query classes and objects.

The EFSOC meta-model of figure 3 was transformed into O-Telos specifications. The O-Telos specifications that result from this process are *meta*-classes. These meta-classes may be instantiated to define specific business processes. E.g., the class `BusinessProcess` may first be defined in Telos as a meta-class, after which it can be instantiated to reflect the business process instance `DisconnectUser` of the case study.

In order to infer validity of access control rules, the content of a ConceptBase repository is subject to queries. The query language is based on deductive rules. In the frame syntax, queries take the following form:

```

QueryClass <query-name> isA <class>
  retrieved_attribute
    <attr-name> : <Class>
    ...
  parameter
    <param-name>: <Class>
    ...
  constraint
    <con-name>: <membership-condition>
end

```

Interpretations of a query are all instances of the super class (`isA`) that fulfill the membership condition. The system will include the retrieved attributes in the answer. If parameters are specified, the user can call a query with values for the parameters. Parameters may appear in the membership condition, which is a logical formula in its own right.

7.1. APPLYING THE EFSOC PROTOTYPE ON THE CASE STUDY

We have used the experimental prototype of the EFSOC framework to facilitate the operational services of the incident handling process. These operational services rely on the EFSOC meta-model that was developed in ConceptBase. ConceptBase proves particularly useful for executing queries for the rule engine. Before querying the repository, however, we firstly need to populate it with instances that were derived from the case study. Below, we have included an example of an instantiation of a Telos class. The frame represents an instance of an access control rule `acr1` which approves that a subject who plays the incident handler role, to execute event operation 3 (send block network address request, see section 5.3.1).

```

AccessControlRule acr1 with
role
  r: IncidentHandlerRole
permission
  p: allow
eventoperation
  o: eo3
end

```

After the EFSOC framework has been populated with the remaining O-Telos frames, representing other constructs such as business process, roles and role assignments, authorization rules may be formulated. These rules can be derived without any loss of semantics from the

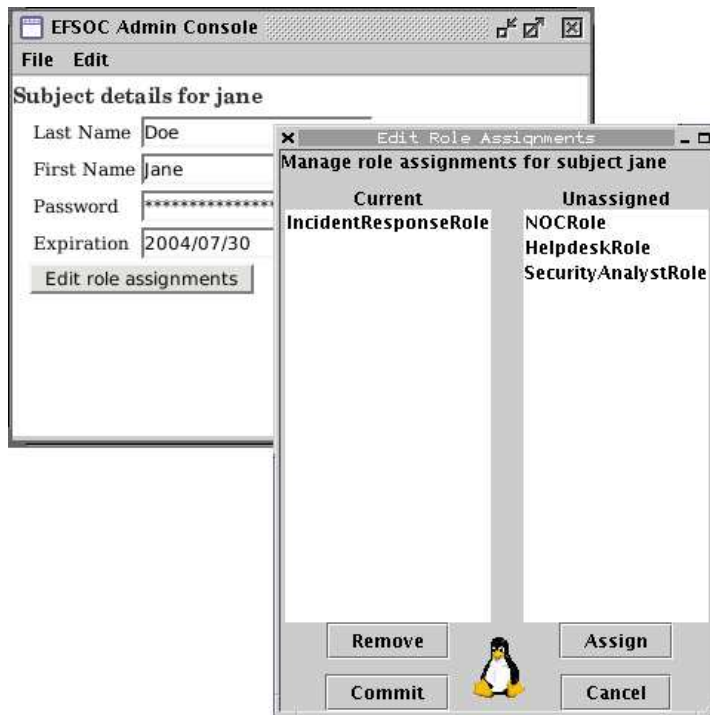


Figure 11. Role Assignment GUI

inferencing rules in first-order predicate logic that were explained in section 6.2. We will now give an example of a (deductive) Telos query.

```
GenericQueryClass RulesApplyingToSubject isA AccessControlRole
with
parameter
    subjectname: string
constraint
    c: $ exists s/Subject ra/RoleAssignment r/Role
        (s name ~subjectname) and
        (ra subject s) and
        (ra role r) and
        (~this role r)
    $
end
```

The above query is defined as a subclass of the `AccessControlRole` class, which signifies that the query returns access control rules. The constraint that is captured by the parameter `c`, is semantically equivalent to its logical counterpart that was given in figure 6.2. Similarly,

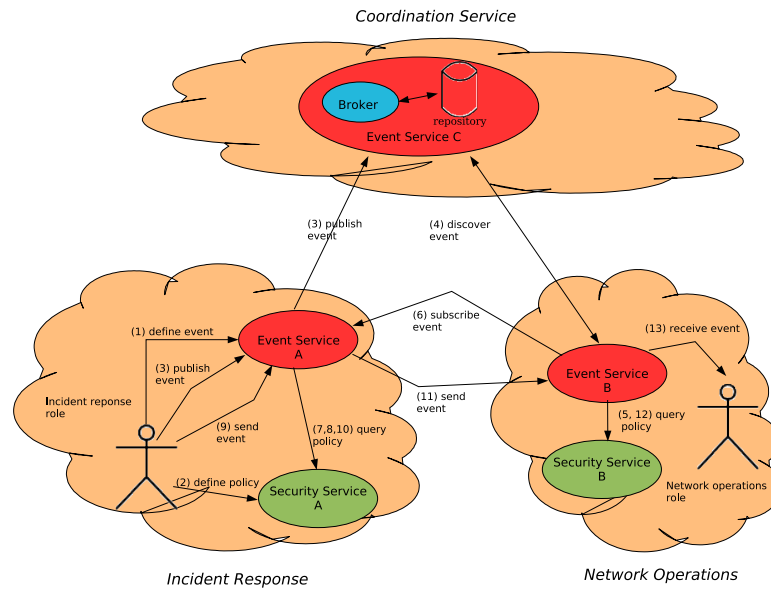


Figure 12. EFSOC infrastructure services in the running example

this query “flows” from one Telos class to the next according to the associations that were laid in the EFSOC metamodel (See figure 3): it firstly “connects” subjects to role assignments, next, resulting role assignments to roles, and finally, these roles to access control rules.

The transformation from EDL fragments into O-Telos frames which will be stored in the ConceptBase system takes place in the repository. For now, these transformation rules are hard-coded in Java. In addition to accessing the repository via the predefined infrastructure services which are primarily meant for interaction between machines, humans-machine interfacing is provided via a GUI application written in Java.

Figure 11 shows a screenshot of the role assignment GUI. On the left hand side, it shows the roles that are currently assigned to subject *jane*, while on the right hand side the unassigned roles are listed.

The EDL definitions may be used to configure infrastructure services in EFSOC. An important feature of EFSOC, is that it may be distributed physically over various distributed nodes. Figure 12 shows how infrastructure services may be distributed over multiple participating nodes in the network. Particularly, each of the nodes runs its own implementation of the EFSOC prototype that was sketched in figure 10, implementing a particular role.

In the context of the running example, this implies that two nodes are introduced: one for the incident handling role and another for the

network operation center. A central broker serves as a UDDI-like registry for advertising and discovering events that may be published or subscribed to by the participating roles. This service may actually be allocated to one of the participants, but in this case we have chosen to implement the service on an individual node stressing the independent and trusted nature of the broker service. As soon as a relevant event is published, the broker service will dispatch it to subscribed participants, after which they may consume the event once it conforms to prescribed security policies.

Within each node, the infrastructure services are thus defined by running an EFSOC prototype. Speaking in terms of the running example this thus means that for both the incident handler and the NOC a instance of an event service and a security service are coded (see figure 12). For reasons of simplicity, the business process service is omitted from the figure.

The protocol for publishing and subscribing events may be organized as a series of thirteen steps. In particular, after a subject, who plays a well-defined role (1) defines an event and (2) defines an access control policy which regulates event operations on the newly defined event, it can be published (3) to the coordinating event service. At this point, other services may discover (4) the event which allows them to determine that event service *A* is the authoritative service for this event. By registering (6) (which is subject to access control rules laid down in security service *B* for the subscribing service (7) and by the access control policies in security service *A* for the providing service (8)), event service *A* knows where to send events when they occur. In addition, event service *A* can now also pro-actively notify its subscribers when access control rules change, or when event definitions are modified.

When an event is generated (9), event service *A* knows who its subscribers are, and, if allowed by security service *A* (10), will propagate this event to event service *B* (11). Event service *B* will query the access control policies in its security service (12) to determine whether it is allowed to receive this event and if so, forward it to the final recipient (13).

It is important to realize that the scenario outlined above demonstrates a typical deployment scenario of the EFSOC framework. The approach outlined here combined role-based thinking with a discretionary approach in the sense that we assume here that the subject who publishes an event also regulates the access to it. In other scenarios, this assumption may not be valid.

8. Conclusions and Future Work

This article has presented a multi-layered, service-based, access-rights framework, named EFSOC, that is grounded on a formal meta-model stating its structural and dynamic characteristics. This framework leverages a set of tailored specification languages for defining and enacting business processes, roles, access control rules, subjects, and events.

EFSOC is an advanced alternative when compared to traditional role-based access control models (RBAC) as, in contrast to RBAC models, it allows to dynamically delegate or retract authorizations to roles. Role assignments are facilitated on an event-driven basis, permitting a push-based web-service interaction protocol. In this protocol, an event may be “pushed” from a request to a subscribed service provider, if the subscriptions comply with prescribed security policies.

Another distinguishing feature of EFSOC is that it addresses security at the level of business processes and authorizations, assuming that message-level security (e.g. non-repudiation, integrity and consistency) is preserved by existing standards, such as WS-Security. At the same time, as EFSOC is standard agnostic, it may be used in combination with existing standard languages, such as WSDL and BPEL.

We believe that the main contribution of this paper lies in the ability of the framework to dynamically reason about (re-)allocations of action authorizations to roles. In addition, other security related definitions in the EFSOC model may be subject to queries. Querying is an important prerequisite for moving towards truly dynamic service interactions. For example, queries may be triggered to find out conflicts between existing authorization-to-role assignments.

The EFSOC framework is validated in two ways. Firstly, the internal validity of the framework is proven by its formalization in O-Telos (and XML-Schema). Secondly, the external validity is addressed by the experimental prototype that supports the specification of EFSOC. This prototype has been successfully applied in the context of a case study, which was described in this paper.

Current work aims at refining and extending the approach in various directions. Firstly, we intend to explore the dynamic nature of event-based business processes further by carefully studying dynamic events. In addition, we wish to examine modifications and maintenance to existing infrastructure and operational services in the EFSOC framework, allowing for pro-active change management support. Lastly, we plan to conduct a series of experiments to explore and further strengthen the EFSOC framework taking into account issues such as delegation of authorizations and conflict resolution.

References

1. Altheim, M.: 2003, 'XML Common Logic (XCL) 1.0'. Technical report, M. Altheim. <http://www.altheim.com/specs/xcl/1.0/>.
2. Andrews, T., F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana.: 2003, 'Business process execution language'. Technical report, BEA Systems and International Business Machines Corporation and Microsoft Corporation and SAP AG and Siebel Systems.
3. Arkin, A.: 2002, 'Web Service Choreography Interface 1.0'. Technical report, BEA Systems, Intalio, SAP, Sun Microsystems. <http://dev2dev.bea.com/techtrack/wsci.jsp> (visited 2-7-2003).
4. Atkinson, B., G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, J. Klein, B. LaMacchia, P. Leach, J. Manfredelli, H. Maruyama, A. Nadalin, N. Nagaratnam, H. Prafullchandra, J. Shewchuk, and D. Simon: 2002, 'Web Services Security (WS-Security)'. Technical report, Microsoft, IBM and Verisign.
5. Botha, R. and J. Eloff: 2001, 'Separation of Duties for Access Control Enforcement in Workflow Environments'. *IBM Systems Journal* **40**(3), 666–682.
6. Box, D., D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer: 2000, 'Simple Object Access Protocol (SOAP) 1.1'. W3C Note, W3C. <http://www.w3.org/TR/SOAP/>.
7. Christensen, E., F. Curbera, G. Meredith, and S. Weerawarana: 2001, 'Web Services Description Language (WSDL) 1.1'. W3C Note, W3C. <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>.
8. Common Logic, 'The Common Logic Standard'. Technical report. <http://cl.tamu.edu/>.
9. Della-Libera, G., B. Dixon, P. Garg, S. Hada, P. Hallam-Baker, M. Hondo, C. K. (Editor), H. Maruyama, A. N. (Editor), N. Nagaratnam, A. Nash, R. Philpott, H. Prafullchandra, J. Shewchuk, D. Simon, E. Waingold, and R. Zolfonoon: 2002, 'Web Services Secure Conversation Language (WS-SecureConversation)'. Technical report, BEA Systems, Inc., Computer Associates International, Inc., International Business Machines Corporation, Layer 7 Technologies, Microsoft Corporation, Netegrity, Inc., Oblix Inc., OpenNetwork Technologies Inc., Ping Identity Corporation, Reactivity, Inc., RSA Security, Inc., Verisign Inc., and Westbridge Technology, Inc.
10. Discretionary Access Control: 1987, 'A Guide To Understanding Discretionary Access Control In Trusted Systems'. Technical Report Library No. S-228,576, National Computer Security Center.
11. Farrell, S., I. Reid, H. Lockhart, D. Orchard, K. Sankar, C. Adams, T. Moses, N. Edwards, J. Pato, B. Blakley, M. Erdos, S. Cantor, R. B. Morgan, M. Chanliau, C. McLaren, C. Knouse, S. Godik, D. Platt, J. Moreh, J. Hodges, and P. Hallam-Baker: 2003, 'Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1'. Committee specification, OASIS. http://www.oasis-open.org/committees/documents.php?wg_abbrev=security.
12. Ferraiolo, D., J. Cugini, and R. Kuhn: 1995, 'Role-Based Access Control: Features and Motivations'. In: *Proceedings of the 11th Annual Computer Security Applications Conference*.
13. Ferraiolo, D. and R. Kuhn: 1992, 'Role-Based Access Control'. In: *Proceedings of the 15th NIST-NSA National Computer Security Conference*.
14. Godik, S. and T. M. (editors): 2003, 'eXtensible Access Control Markup Language (XACML)'. OASIS Standard, OASIS.

15. H.S. Thompson, D. Beech, M. M. and N. M. (editors): 2001, 'XML Schema PART 1: Structures'. Technical report, W3C. <http://www.w3.org/TR/xmlschema-1/>.
16. Jarke, M., R. Gellersdörfer, M. Jeusfeld, and M. Staudt: 1995, 'Conceptbase - a deductive object base for meta data management.'. *Journal of Intelligent Information Systems, Special Issue on Advances in Deductive Object-Oriented Databases* **4**(2), 167 – 192.
17. Leune, K.: 2004a, 'Conceptual overview of the EFSOC event service'. Technical Report 16, Tilburg University, Infolab.
18. Leune, K.: 2004b, 'EFSOC Infrastructure Services'. Technical report, Infolab, Tilburg University, The Netherlands. Available at <http://www.uvt.nl/infolab/report-series/>.
19. Leune, K., W.-J. van den Heuvel, and M. Papazoglou: 2004, 'Exploring a Multi-Faceted Framework for SOC: How to develop secure web-service interactions?'. In: *Proceedings of the 14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government applications*. pp. 56 – 61.
20. Luckham, D.: 2002, *The Power of Events. An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Press.
21. M. Krause, H. T.: 1999, *Handbook of Information Security Management*, No. ISBN 0849398290. Auerbach Publications, 4th edition.
22. Morgan, T.: 2002, *Business Rules and Information Systems*. Addison-Wesley. ISBN 0-201-74391-4.
23. Mylopoulos, J., A. Borgida, and M. Koubarakis: 1990, 'Telos: Representing Knowledge About Information Systems'. *ACM Transactions on Information Systems* **8**(4).
24. of Defense, D.: 1985, 'Trusted Computer System Evaluation Criteria'. Technical Report Library No. S225,711.
25. Papazoglou, M. and G. Georgakopoulos: 2003, 'Introduction to the Special Issue about Service-Oriented Computing'. *Communications of the ACM* **46**(10), 24–29.
26. Pelz, C.: 2003, 'web services orchestration: a review of emerging technologies, tools, and standards'. Technical report, HP.
27. Saltzer, J. and M. Schroeder: 1975, 'The Protection of Information in Computer Systems'. *Proceedings of the IEEE* **63**(9), 1278 – 1308.
28. Sandhu, R., E. Coyne, H. Feinstein, and C. Youman: 1996, 'Role-Based Access Control Models'. *IEEE Computer*.
29. UDDI: 2000, 'Universal Description, Discovery, and Integration (UDDI)'. Technical report, uddi.org. <http://www.uddi.org>.
30. Wahl, M., T. Howes, and S. Kille: 1997, 'Lightweight Directory Access Protocol (v3)'. Technical Report RFC 2251, Critical Angle Inc., Netscape Communications Corp., Isode Limited. <http://www.ietf.org/rfc/rfc2251.txt>.

Appendix

A. XML Schema of EFSOC

```

<?xml version='1.0'?>

<xsd:schema
  targetNamespace="http://www.pronir.nl/efsoc/efsoc.xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xcl = "http://www.altheim.com/specs/xcl/1.0/xcl1.dtd">

  <xsd:element name="efsoc">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="role" minOccurs="0"
          maxOccurs="unbounded" type="roleType"/>
        <xsd:element name="subject" minOccurs="0"
          maxOccurs="unbounded" type="subjectType"/>
        <xsd:element name="roleassignment" minOccurs="0"
          maxOccurs="unbounded" type="roleassignmentType"/>
        <xsd:element name="eventoperationtype" minOccurs="0"
          maxOccurs="unbounded" type="eventoperationtypeType"/>
        <xsd:element name="eventoperation" minOccurs="0"
          maxOccurs="unbounded" type="eventoperationType"/>
        <xsd:element name="eventbody" minOccurs="0"
          maxOccurs="unbounded" type="eventBodyType"/>
        <xsd:element name="permission" minOccurs="0"
          maxOccurs="unbounded" type="permissionType"/>
        <xsd:element name="accesscontrolrule" minOccurs="0"
          maxOccurs="unbounded" type="accesscontrolruleType"/>
        <xsd:element name="accesscontrolpolicy" minOccurs="0"
          maxOccurs="unbounded" type="accesscontrolpolicyType"/>
        <xsd:element name="businessprocess" minOccurs="0"
          maxOccurs="unbounded" type="businessprocessType"/>
        <xsd:element name="query" type="queryType"
          minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:complexType name="roleType">
    <xsd:sequence>
      <xsd:element name="rolename" type="xsd:string"

```

```

        minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID"/>
</xsd:complexType>

<xsd:complexType name="subjectType">
    <xsd:sequence>
        <xsd:element name="subjectname" type="xsd:string"
            minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID"/>
</xsd:complexType>

<xsd:complexType name="roleassignmentType">
    <xsd:sequence>
        <xsd:element name="subjectid" type="xsd:QName"/>
        <xsd:element name="roleid" type="xsd:QName"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID"/>
</xsd:complexType>

<xsd:complexType name="eventoperationtypeType">
    <xsd:sequence>
        <xsd:element name="eventoperationtypename" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID"/>
</xsd:complexType>

<xsd:complexType name="eventoperationType">
    <xsd:sequence>
        <xsd:element name="eventoperationtypeid" type="xsd:QName"/>
        <xsd:element name="eventid" type="xsd:QName"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID"/>
</xsd:complexType>

<xsd:complexType name="eventType">
    <xsd:sequence>
        <xsd:element name="envelope" minOccurs="1"
            type="EventEnvelopeType" maxOccurs="1"/>
        <xsd:element name="body" minOccurs="0"
            type="EventBody" maxOccurs="1"/>
    </xsd:sequence>

```

```

    <xsd:attribute name="id" type="xsd:ID"/>
  </xsd:complexType>

  <xsd:complexType name="EventEnvelopeType">
    <xsd:sequence>
      <xsd:element name="eventtype" type="EventType"
        minOccurs="1" maxOccurs="1"/>
      <xsd:element name="eventid" type="EventIDType"
        minOccurs="1" maxOccurs="1"/>
      <xsd:element name="timestamp" type="TimeStampType"
        minOccurs="1" maxOccurs="1"/>
      <xsd:element name="reference" type="ReferenceType"
        minOccurs="0" maxOccurs="1"/>
      <xsd:element name="sender" type="SubjectID"
        minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:simpleType name="EventType">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>

  <xsd:simpleType name="EventIDType">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>

  <xsd:simpleType name="TimeStampType">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>

  <xsd:simpleType name="SubjectID">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>

  <xsd:simpleType name="ReferenceType">
    <xsd:list>
      <xsd:simpleType>
        <xsd:restriction base="xsd:string"/>
      </xsd:simpleType>
    </xsd:list>
  </xsd:simpleType>

  <xsd:complexType name="EventBody">

```

```

    <xsd:simpleContent>
      <xsd:restriction base="eventBodyType">
        <xsd:attribute name="id" type="xsd:ID"/>
      </xsd:restriction>
    </xsd:simpleContent>
  </xsd:complexType>

<xsd:complexType name="eventBodyType">
  <xsd:simpleContent>
    <xsd:restriction base="xsd:schema">
      <xsd:attribute name="id" type="xsd:ID"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="permissionType">
  <xsd:simpleContent>
    <xsd:restriction base="xsd:string">
      <xsd:attribute name="id" type="xsd:ID"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="accesscontrolruleType">
  <xsd:sequence>
    <xsd:element name="permissionid" type="xsd:QName"/>
    <xsd:element name="eventoperationid" type="xsd:QName"/>
    <xsd:element name="roleid" type="xsd:QName"/>
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:ID"/>
</xsd:complexType>

<xsd:complexType name="accesscontrolpolicyType">
  <xsd:sequence>
    <xsd:element name="eventid" type="xsd:QName"/>
    <xsd:element name="accesscontrolruleid" type="xsd:QName"
      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:ID"/>
</xsd:complexType>

<xsd:complexType name="businessprocessType">
  <xsd:simpleContent>

```

```

        <xsd:restriction base="xsd:sequence">
            <xsd:attribute name="id" type="xsd:ID"/>
        </xsd:restriction>
    </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="queryType">
    <xsd:sequence>
        <xsd:element name="parameter" type="xsd:string"
            minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="formula" type="xcl:formula"
            minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>

</xsd:schema>

```

B. EDL query definition example

```

<efsoc:efsoc
  xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
  xmlns:xcl = "http://www.altheim.com/specs/xcl/1.0/xcl1.dtd"
  xmlns:efsoc = "http://www.leune.org/tmp/efsoc.xsd">

  <efsoc:query name="rulesForSubject">
    <efsoc:parameter name="input_name" type="xsd:string"/>
    <efsoc:formula>
      <xcl:conn name="implies">
        <xcl:quant name="exists" variable="s">
          <xcl:type>Subject</xcl:type>
        <xcl:quant name="exists" variable="ra">
          <xcl:type>RoleAssignment</xcl:type>
        <xcl:quant name="exists" variable="r">
          <xcl:type>Role</xcl:type>
        <xcl:quant name="exists" variable="ru">
          <xcl:type>AccessControlRule</xcl:type>
        <xcl:conn name="and">
          <xcl:pred name="name">
            <xcl:term name="s"/>
            <xcl:term name="input_name"/>
          </xcl:pred>
        </xcl:conn>
      </xcl:conn>
    </efsoc:formula>
  </efsoc:query>

```

```
<xcl:pred name="subject">
  <xcl:term name="ra"/>
  <xcl:term name="s"/>
</xcl:pred>
<xcl:pred name="role">
  <xcl:term name="ra"/>
  <xcl:term name="r"/>
</xcl:pred>
<xcl:pred name="role">
  <xcl:term name="ru"/>
  <xcl:term name="r"/>
</xcl:pred>
</xcl:conn>
</xcl:quant>
</xcl:quant>
</xcl:quant>
</xcl:quant>
<xcl:pred name="result">
  <xcl:term name="s"/>
</xcl:pred>
</xcl:conn>
</efsoc:formula>
</efsoc:query>
</efsoc:efsoc>
```

