Vrije Universiteit Brussel

FACULTY OF APPLIED SCIENCES

ACCESS CONTROL CSL &

CSL2PADUS TRANSFORM FRAMEWORK

Shaonan WANG

Academic Year: 2006- 2007

Promoter: Prof. Viviane Jonckers

Copromotor: Dr. Wim Vanderperren

Niels Joncheere

Thesis submitted in partial fulfillment of the requirements for the Master's degree of Applied Computer Science.

Abstract

The traditional workflow and business process modeling languages suffer from two problems in providing users an easy to use environment: lack of support for a visual creation environment and poor modularization of crosscutting concerns.

At the SSEL lab, two approaches are proposed to address the identified problems in current workflow languages, namely Padus and the SCE.

Padus is an aspect-oriented extension for BPEL in order to modularize crosscutting concerns in WSBPEL.

The Service Composition Environment (SCE) provides a plug-and-play service composition environment that guides the user to a valid composition.

However, Padus aspects are still relatively low-level and are difficult to develop and understand for non-programmers. Therefore, we propose to use higher-level concern specific languages (CSL) that each target one concern.

As an example CSL, we propose a role based access control language.

Additionally, we introduce a generic transformation framework to transform any CSL program into executable Padus aspects and we provide an example application of the framework for the role based access control CSL.

Acknowledgement

This thesis is a big challenge for me, and without the help from the following people, it would be impossible to be finished. I'd like to say thank you to:

- Prof. Dr. Viviane Jonckers, from whom I learn both the knowledge and the spirit of computer science;
- Dr. Wim Vanderperren, for giving me the opportunity to do my thesis in SSEL lab, and leading me to access the research world;
- Niels Joncheere, for his detailed explanations for specific technology questions I've met;
- Other members in SSEL lab, for their advises whenever I need help;

And after all, I should thank my father, for his unlimited love and support.

Table of Contents

List of Figures

List of Code Fragments

1 Introduction

1.1 Motivation

Over the last years, web services (1) have gained a lot of popularity in both academics and industry as a technology that promotes the reuse of software applications at the service level, independent of underlying programming languages and system platforms. For example, web services enable a C# application based on the .NET platform to communicate with a Java application based on the J2EE platform.

The true value of web services lies in the dynamic cooperation of individual web services. Current workflow or business process modeling technologies (2) involve two styles of web service cooperation: orchestration and choreography. The orchestration style composes existing web services in order to achieve more advanced functionality and exposes the resulting composition as a new service, which can be composed recursively. The term orchestration refers to the fact that the cooperation of the different services is governed by a central composition, similar to the role of a conductor in a symphonic orchestra. On the other hand, the choreography style describes a predefined peer-to-peer protocol between the different services. All services are equal and communicate with each other in accordance with the protocol. The term choreography stems from dancing terminology, where it implies that dancers dance while adhering to a global scenario without a single point of control.

BPEL (3) is currently the de facto standard for web service orchestration, and is based on the XML and WSDL standards. It provides a small but powerful

language for process-oriented programming. There are, however, two main drawbacks from a usability perspective.

The first drawback is that BPEL suffers from poor separation of concerns: each process is implemented using one monolithic specification, and certain concerns of the process do not align with the main logic of the process, but end up scattered across the process and tangled with one another. This makes it difficult to add, remove or maintain these concerns. These concerns are called crosscutting concerns, and typical examples include access control and billing. Because of this drawback, processes become complex and inflexible, while fierce market competition increases the need for agile solutions.

The second drawback is that BPEL does not have a standardized visual notation. BPEL is merely an XML language and requires a certain amount of in-depth knowledge in order to compose web services. This makes it difficult to react to changing markets because business experts are are not sufficiently involved in business process modeling.

Previous research (4) (5) (6) (7) has applied aspect-oriented programming to workflow languages in order to improve separation of concerns in business process modeling. Aspect-oriented languages can be classified into two categories according to the way in which crosscutting concerns are applied to the application. Dynamic languages can add, remove or modify crosscutting concerns at runtime, but require a dedicated runtime platform and are thus less compatible with the existing tool chain. Additionally, dynamic languages tend to have more runtime overhead. On the other hand, static languages operate at compile time, and require restarting the application each time a concern is added, removed or modified. They typically have less runtime overhead. The ideal choice between these two categories depends on the specific application requirements.

Padus (4) is a static aspect-oriented language for BPEL that allows crosscutting concerns (such as access control and billing) to be specified separate from the main control flow of the process. In order to facilitate web service composition by developers that lack indepth knowledge of BPEL or Padus, a visual Service Creation Environment (SCE) (8) has been proposed which allows expressing concerns using dedicated Concern Specific Languages (CSLs). The SCE provides a graphical interface for expressing web service compositions while supporting modularization of crosscutting concerns. The SCE consists of three main components: a services repository that contains the web services that can be composed, a composition templates repository that contains a set of abstract web service compositions, and a crosscutting concerns repository that contains the Padus or CSL implementations of a set of crosscutting concerns. New web service compositions can be created by visually connecting elements from these repositories on a canvas.

This thesis proposes a role based access control CSL that aims to address process level access control in the telecom community. In order to facilitate the integration of CSLs into the SCE, this thesis also proposes a CSL to Padus aspect transformation framework implemented using Java.

The structure of this thesis is as follows: chapter 2 introduces the research context of this thesis, including web services technology, workflow and business process modeling, aspect-oriented programming, aspect-oriented workflow languages, Padus, CSLs and the SCE. Chapter 3 presents the design and implementation of our role based access control CSL. Chapter 4 presents the CSL to Padus transformation framework, and chapter 5 states our conclusions.

2 Research Context

2.1 Web Services Technologies

2.1.1 What are web services?

(9) defines a web service as a software system designed to support interoperable machine-to-machine interaction over a network. In other words, web services allow applications from different platforms, written in different programming language to communicate with each other.

2.1.2 How do web services work?

2.1.2.1 XML and web services

Web services communicate with each other using XML message by interrelated protocols. XML itself is used to describe data in a platform, language independent way. As a result, web services, which building on top of XML, provide a protocol framework to connect service providers and consumers of different platforms.

2.1.2.2 Web service stack

Figure 1 web services stack

The interrelated modular protocols standardize all aspects of the web services XML message communication, ranging from messaging, service description, publication, discovery to security, reliability, business processes, etc...Figure 1 web services stack, adopted from (10), is a web services stack that illustrates the web service specification framework. Messaging protocol, service description protocol, publication and discovery protocol form the foundation layer of web service framework. More advance protocols are built on top of the foundation layer, to standardize extra aspects of web services.

2.1.3 Foundation protocols: SOAP, WSDL, UDDI

While this thesis is written, higher level protocols in the web services stack are still being proposed, some proposals are even competing with each other. To improve the web services interoperability, Web Services Interoperability Organization (WS-I) (11) publishes WS-I profiles to guide developing interoperable web services. A profile is a set of specifications at a specific level with guidelines and conventions for using the specifications together. WS-I basic

profile 1.0 (12) includes SOAP 1.1, WSDL 1.1, UDDI 2.0, XML 1.0, XML Schema and HTTP 1.1. Even thought there are alternatives protocols, SOAP, WSDL and UDDI are widely accepted in industry, SOAP and WSDL are even mandatory specifications in WS-I.

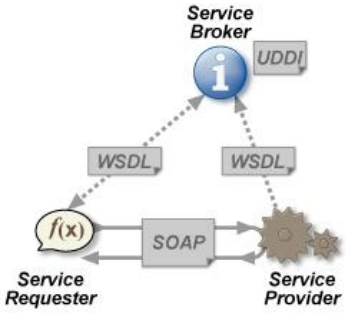

Figure 2 Web Service Core Specifications

SOAP (13) is the foundation layer protocol to transport XML-based messages through the internet, and it's usually bound to http protocol. Web Services Description Language (WSDL) (14) is an XML based language which provides a service level interface to describe the functions this service without underlying implementation information. UDDI (15) is a service broker where services providers can register themselves in and services consumers can find services they need. Figure 2 Web Service Core Specifications (16) shows how SOAP, WSDL and UDDI can cooperate with each other to achieve the interoperability between applications of different platform or programming language. Service providers register to UDDI with description of their serivces in the form of WSDL, while service rquesters send their require of the services also in the form of WSDL. When UDDI finds a matching service provider, UDDI will send the matching WSDL to the service requester. With the information provided by matching WSDL, the service requester can begin to communicate with the service provider

directly with SOAP protocol. The following sections give more detailed introduction to most widely accepted SOAP and WSDL specifications.

2.1.4 SOAP

Figure 3 SOAP Envelope

SOAP, ever known as simple object access protocol, later as service oriented architecture protocol, and now just as SOAP without any acronym meaning, provides the definition of the XML-based information which can be used for exchanging structured and typed information between peers in a decentralized, distributed environment. As shown in Figure 3 SOAP Envelope, a soap message wraps the XML message to be sent into an envelope with a header with additional information like context, authentication, and management. Even though SOAP is commonly bound to HTTP, SOAP is independent of underlying transport protocols. It can also be bound to SMTP or XMPP for example.

2.1.5 Web service description language (WSDL)

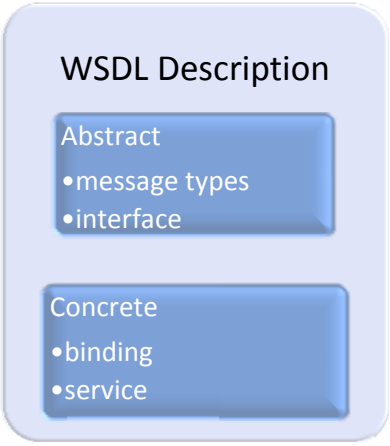

Figure 4 WSDL Component

Given SOAP provides a message communicating structure between web services, web service description language (WSDL) describe the web services itself. Since WSDL version 1.1 has not been endorsed by W3C, while version 2.0 is a recommendation, we will follow the syntax of WSDL version 2.0 in the rest of this section.

To improve the reusability, a WSDL description consists of abstract component to describe the reusable part of a service description and concrete component to provide service specific information.

As shown in Figure 4 WSDL Component, the abstract components include message types and interface. Message type's component describes the kind of messages the service will send and receive; interface describes what abstract functions service will provide.

The concrete description is described by binding and service component. Binding specifies how to access the service, such as the protocol to transport message,

which is usually SOAP, and also the concrete message type. Service component describes where to access the service, most importantly, physical address of the service.

There are also other components besides the above mentioned elements, for example, documentation component provides more application-level requirements for the use of the service; fault component is used to describe the exceptional situation; import and include component help to promote reuse of service descriptions. For more detailed information about WSDL version 2, please refer to (14).

2.2 Workflow Languages

2.2.1 Business Process Modeling

To understand business process modeling, we need first to understand what a business process is. A business process is a set of interrelated tasks to fulfil a goal in an enterprise, usually with well-defined inputs and outputs. A business process describes what happens and how it happens in an enterprise activity. Business process modeling is to provide a type level description of the business processes of the same nature, so that business analysts and managers can study the business processes to improve the efficiency of enterprise activities. Workflow is a closely related concept to business process modeling. Workflow has existed decades before the term of business process modeling comes into exist. Workflow is used to refer to the subject now is called business process modeling. There is a subtle difference between workflow and BPM. According to (2), workflow is an IT technology that uses electronic systems to manage and monitor business processes.

Either BPM or Workflow software, the two main purposes are to model the actual enterprise activities, and to change the processes of the activities to be more efficient and put the changes into practice. Firstly, modeling the business processes involves presenting the activities in an enterprise as loosely couple services, and linking the services together to achieve more advanced functions. With the wide acceptance of Web Services standards in industry, service oriented architecture has gained a lot of popularities recently. Secondly, improving business processes and putting the change into practice also become much easier with Web Services. Business Process Executable Language (BPEL) is language based on web services technology that can generate business process fully

executable. The following sections will first introduce service-oriented architecture, and then explain BPEL language in more detail.

2.3 Service-Oriented Architectures: Orchestration and Choreography

Service oriented architecture (SOA) is a natural evolution of software engineering development. During the last few decades, with the development of software engineering, the granularity of software reusability has kept increasing from procedure-oriented, object-oriented to component-oriented. SOA provides an even more dynamic and flexible reuse solution by separating concerns into loosely couple services and standardized interfaces for the services to communicate with each other without the foreknowledge of the underlying platform and programming language. In a service oriented architecture, a service written in Java in J2EE platform can consume a service written in C# in .Net platform.

SOA is an information systems architecture style that doesn't constrain to a specific implementation technologies. Although REST, RPC, DCOM, CORBA Web Services and WCF can all implement SOA, the advent of Web Services has greatly popularized SOA, and Web Services is the most widely accepted technology to implement an SOA.

There are two structural styles to link services together in SOA: Orchestration and Choreography.

In Orchestration composition style, there is a central control service which gives orders to the other services about what to do and when to do it, just like a conductor in an orchestra; choreography style provides a contract about message

exchanging ordering rules which all the services must follow to communicate with each other, just like dancers with a choreography which has been agreed upon before.

The main characters of orchestration are:

1. Executable: Orchestration tells the services exactly what to do, so orchestration workflow languages are usually executable.
2. Recursive composition: Orchestration is often referred to as a recursive composition of services, since it's all about composition the existing services and exposing the result as a new service.
3. Business Process Execution Language (BPEL) (3): BPEL is the de facto language in industry to implement orchestration style SOA.

The main features of choreography are:

1. Descriptive: Choreography considers the communications between services in terms of observable messages, and it describes the ordering rules for the messages. Although not executable, with all the services follow the same ordering rules, choreography can generate business process depending on the dynamic situation of services communication.
2. Peer to peer collaboration: all the services respect the same contract to communicate with each other, there is no a super service which has more power than the others.
3. Web Services Choreography Description Language (WS-CDL) (17): WS-CDL is the state of art choreography language.

The relationship between orchestration and choreography is complementing rather than competing. (18) has pointed out that one can realize an SOA with WS-CDL

as a blueprint to link islands of orchestration BPEL processes. Even though both BPEL and WS-CDL are of their value, for the timing being of writing this thesis, BPEL is still de facto workflow language to implement SOA. Our research is based on BPEL, so the next section will introduce more details of BPEL.

2.3.1 WS-BPEL

BPEL is an orchestration language based on XML and WSDL to model executable business processes by composing existing services into a more advanced service. BPEL models business processes from the perspective of web services composition through a set of well- defined activities. There are two categories of activities in BPEL: basic and structural. Basic activities are elementary steps of process behavior, while structured activities model control-flow logic that composes the basic activities or other structured activities.

Some examples of the basic activities as explained in (3) are:

- <invoke> activity is typically used to invoke an operation of a service described with WSDL.
- <recieve> and <reply> activities are used to provide services to the service consumer.
- <throw> activity is used to specify an internal fault explicitly.
- <wait> activity specifies a delay for a certain period of time or until a certain deadline is reached.
- <exit> activity is used to immediately end the business process instance.

Examples of structured activities of BPEL from (3) are:

- <sequence> activity contains one or more activities that are performed sequentially, in the lexical order in which they appear within the <sequence> element.
- <if> activity provides conditional behavior.
- <while> activity provides for repeated execution of a contained activity.
- <repeatUntil> activity provides for repeated execution of a contained activity.
- <pick> activity waits for the occurrence of exactly one event from a set of events, then executes the activity associated with that event.
- <flow> activity provides concurrency and synchronization.
- <forEach> activity will execute its contained <scope> activity exactly N+1 times where N equals the <finalCounterValue> minus the <startCounterValue>.

Besides the activities, BPEL also has other notation to facilitate modeling business processes, such as partner link type, partner link and endpoint to model the relationship between two communicating services; variable and message for data handling; scope for compensation handler, fault handler, termination handler and event handler, etc...

While BPEL plays the role of de facto technology to model business process in industry, Business Process Modeling Notation (BPMN) (19) has been proposed. Rather than modeling business process in an XML based language, which requires in-depth knowledge about the related technology before developing real application as BPEL does, BPMN is a standardized graphical notation to draw business processes in a workflow. Even though BPMN is intended to cooperate with BPEL to provide a user-friendly interface to model business processes, the fundamental difference underlying the two technologies makes it very difficult to

convert BPMN diagrams to human readable BPEL processes. BPMN is in fact also helpful to our research, since we provide a visual service creation environment similar to BPMN diagrams. More introductions to the visual service creation environment will be given in later sections.

2.4 Aspect-Oriented Programming

2.4.1 Motivation of AOP

Aspect-oriented programming (AOP) is programming paradigm in software engineering to address the modularization of crosscutting concerns in object-oriented software development.

Object-oriented paradigm contributes to separating concerns by modularize different concerns in entities like packages, classes and methods, thus greatly improve the reusability of applications and develop efficiency, since when one concern needs changing, only the corresponding application entity needs to be changed while the other entities can remain the same.

While OO concept has greatly improve the reusability of software applications, not all concerns can be modularized in one entity within the object-oriented paradigm. Logging is a typical example of crosscutting concern, which entails scattering all over the classes that need to be logged while the other concerns are tangled with logging concern. A change in logging concern leads to change all the classes to be logged, and when other concerns need changing, developer must understand the logging concern first to make the change. Billing and access control are also examples of crosscutting concerns: billing in telecommunication industry is often tangled with other business logic concerns such as pricing policy, service duration and the customer allocation; access control is a security concern scattered application systems to make sure the user has the authentication before confidential operations are executed.

In object-oriented paradigm, crosscutting concerns have greatly increased the complexity of software applications and made changes difficult to implement. Aspect-Oriented Programming complements object oriented programming by providing a new construct to modularize crosscutting concerns called aspect, working together with classes in object-oriented paradigm to modularize normal concern, thus improves the separating of concerns in software engineering development.

AspectJ is an aspect oriented extension language for JAVA, and it's also the most popular general-purpose AOP language. The following sections will take AspectJ as an example to introduce AOP language in more details and then briefly introduce aspect language implementation technologies: aspect weaving.

2.4.2 AspectJ

Aspect-Oriented Programming language applies joint point model to crosscutting concern by adding extra behavior at certain points of the base object oriented program. Three main elements of a join point model are joinpoint, pointcut and advice.

The certain execution point in the base program where extra behavior is added is called a join point. AspectJ join point model provides a variety of join points such as method or constructor call or execution, the initialization of a class or object, field read and write access, exception handlers.

Pointcut is a set of join points defined by pointcut designators. A pointcut defines when the extra behavior in an aspect should happen. Some examples of AspectJ pointcut from (20) are as following:

- `get(Signature)` every reference to any field matching `Signature`
- `this(Type or Id)` every join point when the currently executing object is an instance of `Type` or `Id`'s type
- `cflow(Pointcut)` every join point in the control flow of each join point `P` picked out by `Pointcut`, including `P` itself
- `Pointcut0 && Pointcut1` each join point picked out by both `Pointcut0` and `Pointcut1`

While a pointcut defines when extra behavior should happen, an advice defines what the extra behavior is. There are three kinds of advices in AspectJ: before, after and around. As suggested by the names, an advice adds extra behavior before or after the joinpoints picked up by the pointcut, or in case of around, replaces the original behavior.

Besides join point model, AspectJ also provides inter-type declarations which allow aspect define new members within other classes. (20) shows an example how to use inter-type declaration in AspectJ:

```
  aspect A {
    private interface HasName {}
    declare parents: (Point || Line || Square)
implements HasName;

    private String HasName.name;
    public String HasName.getName() { return name; }
  }
```

Code 1 An aspect in AspectJ

2.4.3 Aspect Weaving

Aspect weaving refers to applying aspects to bases classes to realize the crosscutting behaviors addressed by the aspects. Static weaving and dynamic weaving (21) are the two approach of aspect weaving.

Static weaving applies aspects to the base program at compile time or load time. This approach firstly identifies all the possible join points at compile time, and then either adds the advice functions to these join point at compile time or at load time if the advice behavior depends on the information available only at load time. AspectJ is an example language implemented by static weaving. Static weaving provides a speed performance comparable to the traditional. Static weaving requires no changing in Java virtual machine, thus produces speed performance comparable with that of the original application, and good compatibility with the platform and environment of the base language. The disadvantage is that since all aspect related behavior is integrated into the base program before runtime, crosscutting functionalities and normal functionalities cannot be identified dynamically while the application is running. This disadvantage greatly restricts the AOP applications in the run-time and long running systems.

Dynamic weaving affects the base program at runtime by adapting the virtual machine to be aspect aware. With this approach, aspects can be woven or unwoven at runtime, thus makes changing crosscutting behavior on the fly possible. Several dynamic AOP languages have been proposed like PROSE (22)and JAsCo (23). Even though dynamic AOP supports woven, unwoven and replace AOP at runtime, there are some side effects like performance overhead, insecurity and compatibility with the base language platform. Insecurity issue concerns about the possibility to add malicious advices at runtime. Dynamic

weaving decreases the compatibility of the AOP application because it’s unavoidable to update the virtual machine to be aspect-aware.

2.5 AOP and Workflow Languages

2.5.1 Motivation

With widely acceptance of web services technology and Service Oriented Architecture to implement enterprise business process management systems, there is rising demand to improve the existing workflow language. Two problems of current solution for services composition are static selection of web services and poor modularizing crosscutting concerns in procedural oriented programming.

Web services and SOA advocates service level programming independent of the underlying software platform or programming language. Web Services describe the services with the uniform interface, WSDL, and orchestration workflow languages like BPEL compose the interfaces of the services to produce more advanced functionality. Since the BPEL communicates with web services through WSDL interfaces, any services implementing the interface can be used to realize SOA business process. There can be different services implement the same interface, BPEL uses static binding to choose which implementation for a service interface, which means the process needs stopping when change to another implementation, thus it's unaffordable for long running process to change the binding services, even though in real world, new services are published and odd ones disappear quite often.

Another problem in current service composition is poor modularizing crosscutting concerns in orchestration services composition. Two subclasses of crosscutting concern have been identified: procedural level and service level. The original AOP research is focused on object-oriented paradigm where crosscutting concerns tangled and scattering all over the normal concerns. Procedural-oriented

programming, however, also suffers from the crosscutting concerns. For instance, in the case of telecom system, nearly every process begins with an authentication activity to check if the customer has the right to invoke the following service, thus the authentication concern are scattered all over the application; while billing concern frequently involves check the duration of the usage of certain service, and the pricing policies, thus billing is tangled with many other concerns in the system. Service level crosscutting concerns involves the crosscutting concerns generic to all the services like billing, transaction, selection, and caching.

2.5.2 Aspect-Oriented workflow languages

To address the above mentioned problems, several AOP approaches have been proposed including Padus (4), WSML (7), AO4BPEL (5) and Towards Aspect Weaving Applications (6). Padus, which is the base of this thesis work, is an aspect-oriented workflow language addressing the process-level crosscutting concern. As a static AOP approach, an obvious advantage of Padus is good compatibility with the existing tool chains. The following section will give a detailed introduction to Padus. Web service management layer (WSML) is a dynamic aspect-oriented middleware framework between the web services and web services composition client. WSML addresses the service-level crosscutting concerns and just-in-time web service selection and integration. WSML and Padus are complimentary with each other. AO4BPEL is a dynamic aspect-oriented extension for BPEL trying to address all the problems above. Besides the compatibility limit which requires an aspect-aware BPEL engine, the lower level pointcut described by Xpath language is also less expressive compared with the dedicated logic pointcut language of Padus.

2.5.3 Padus

Padus is an aspect-oriented extension for BPEL to address the procedural level crosscutting concerns especially in the context of Telecom Company. Given the requirement of speed performance and compatibility with the existing tool chains, Padus takes a static approach for weaving. For the join point model, Padus applies logic meta-programming (24) (25) to the pointcut language.

Corresponding to the two categories of activities of BPEL, there are two kinds of joinpoint in Padus: behavioral joinpoints and structural joinpoints to identify the particular execution point in a BPEL process. For example, the “invoking” behavioral joinpoint aims to identify the “invoke” activity in BPEL process. The properties of a joinpoint are associated with the attributes or elements of the corresponding BPEL activity. For example, in Padus, the “invoking” joinpoint can have properties such as name, partnerLink, prototype, operation, input Variable, outputVariable to recognize the “invoke” activity of BPEL with the corresponding attributes.

The pointcut language specifies a collection of joinpoints where advance shall be applied by either binding the attributes of the joinpoint with concrete predicates of the desired execution points, or restricting additional properties of joinpoints such as the process or process instance a joinpoint occurs in.

The advice language of Padus can insert BPEL activities into the execution point of the base process specified by the pointcut language. Beside the traditional before, after and around advice, Padus also defines an “in” construct to add additional behavior such as a concurrent activity in a flow activity. Even though the in construct can be realized by around advice sometimes, this will produce significant code duplication.

Padus aspect module includes the above mentioned pointcut language to identify where in the core processes changes need to happen, and advice language to specify what change should be made. In addition, there are also infrastructures to facilitate reuse the aspect definition, such as “using” declaration and abstract advice, pointcut definitions.

Aspect deployment language of Padus has two functions: aspect instantiation and aspect composition. Aspect instantiation defines the Padus aspects should apply on which base processes, while aspect composition defines the order priority of the different aspects if this aspect should apply to the same joinpoint in the execution.

With the above described Padus language, the procedural level crosscutting concern in the orchestration business process modeling can be effectively modularized, thus, the complexity of the business process model can be significantly decreased, and the IT infrastructure become more agile to the ever changing demand from the fierce market competition. However, the trade-off of the advantage of Padus is that the business process developers need an in-depth knowledge of Padus as well as the underlying BPEL processes to make any change on the business process model. This is contradicting to our goal to be easy to maintain and user friendly. The following sections will introduce the complementary researches based on Padus to achieve a more user-friendly interface to apply Padus into practice.

2.5.4 Concern Specific Language

As explained in previous section, Padus is a general-purpose aspect-oriented workflow language, which means Padus is a powerful expressive language capable of dealing with generic problems in the context of procedural level

crosscutting concerns. However, to fit Padus better in the business process modeling context where easy to use and agile to the ever-changing market environment is the essential element, concern specific languages (CSL) based on Padus are proposed to assist constructing a user-friendly process composition environment.

According to the scope of problems a programming language aims to solve, we can categorize two classes of languages: general-purpose language and domain specific language (26). General-purpose language provides a general solution to a wide range of problems in certain area, such as Java and UML. However, this kind of solution may be suboptimal. Domain specific language focuses on a specific domain, and provides more powerful tools to solve a specific problem. (27) gives a vivid comparison that a domain specific language is like a drill, which is a powerful tool to perform a variety of tasks, in the context of putting holes into somewhere. A general-purpose language is like a workbench with tools to perform a variety of tasks. Programmers who are working at their workbench may find a domain specific language fits exactly to the task he or she is working at.

Concern specific language is the domain specific language for Padus. In support for each crosscutting concern in development, there is a concern specific language developed. (8) proposes a billing concern specific language to deal with the billing crosscutting concerns in the context of service delivery platform (SDP) in the telecom community.

```
<concern language="billing" type="time" name="billcall">

    <!-- specify when billing should occur: -->
    <start when="invoking(Service, Port, 'connect', User)" />
    <end when="invoking(Service, Port2, 'disconnect', User)" />

    <!-- specify what should be charged: -->
    <advice>
        <begin> <charge type="setup" context="User" /> </begin>
        <success> <charge type="time" context="User, $Time" /> </success>
        <fail> <!-- do nothing --> </fail>
        <finally> <!-- do nothing --> </finally>
    </advice>
</concern>
```

Code 2 Billing CSL example

The billing CSL is an XML based language addressing the billing crosscutting concern. It identifies two essential elements in billing activities: when billing happens and what should be charged. It detects the operations that should be charged and then sends the detected information as well as the timestamp to a dedicated charging service. The charging service will keep a complete log of all charged events, which will be collected later for generating bills for the customer, possibly affected by business rules. As shown in Code 2 Billing CSL example adopted from (8), user can define the billing activity straightforwardly without the profound knowledge of Padus or even BPEL.

Concern specific languages for Padus are developed in an ad hoc manner. Next chapter will introduce an access control concern specific language. Before that, next section will introduce a visualized service creation environment where both traditional WSDL and CSL can be composed visually by dragging and dropping.

2.5.5 Visual Web Service Creation Environment

Visual web service creation environment (SCE) provides an even higher level abstraction than work flow language such as BPEL to compose web services. In addition, SCE also supports modularizing crosscutting concerns both by Padus and

CSL. There are three kinds of repositories in SCE: WSDL-documented services, composition templates specified in BPELs, and crosscutting concerns either as Padus aspects or CSL programs. When use needs to compose web services into a new business process, it suffices to choose a composition template with place holders for the component web services, and then fill in the place holders with concrete web services by dragging and dropping WSDL services from the service repository. If crosscutting concerns are involved, user can either add a Padus aspect to the related service as in Figure 5 Screenshot of the SCE's interface, or add a CSL as in

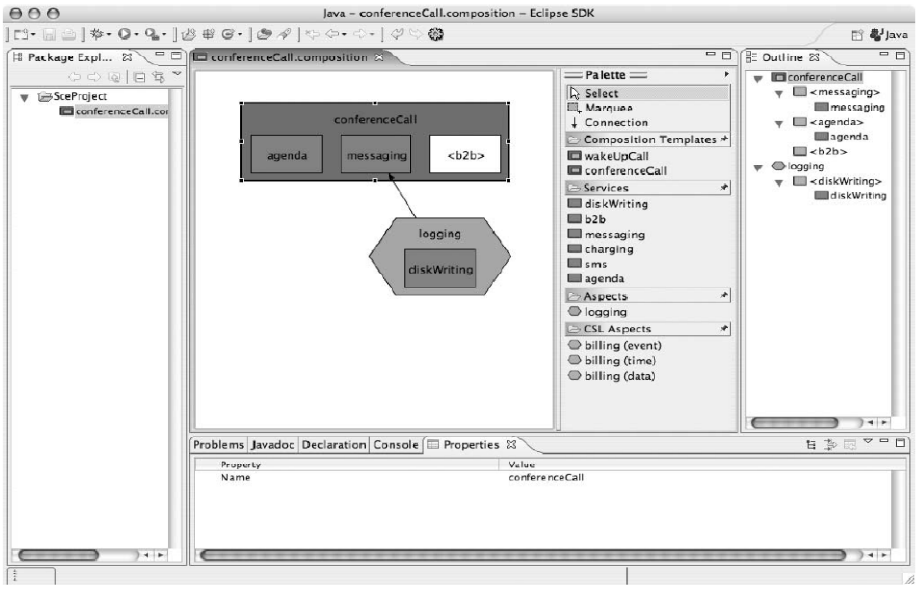

Figure 5 Screenshot of the SCE's interface

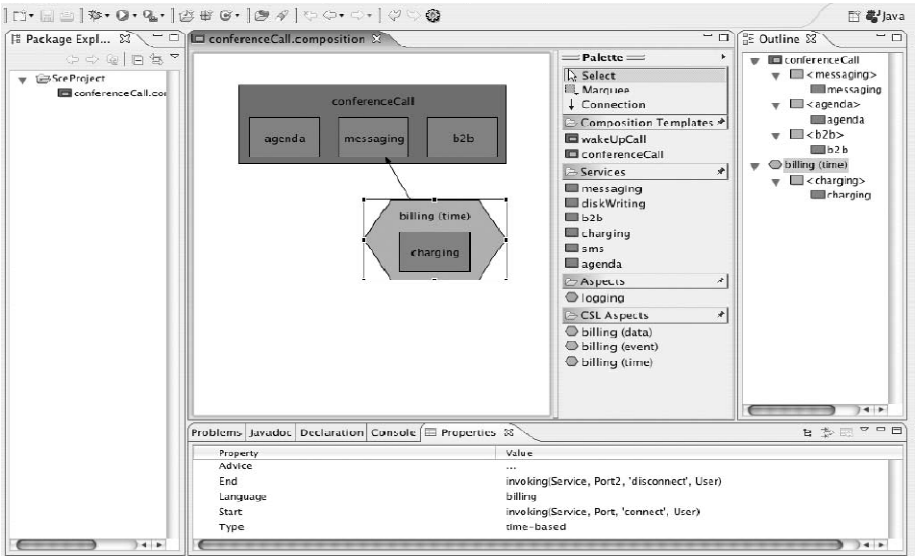

Figure 6 A composition with a concern-specific language

When a user composes services in SCE, the compatibility between the services and the composition template is checked automatically. SCE will generate a report with the mismatch feedback to the user, if the service turns out to be in compatible.

SCE handles CSL by first transforming CSL to Padus aspects. Chapter 5 introduces a transforming framework to assist SCE implementation.

3 Access Control Concern Specific Language

3.1 Design of Enterprise Role Based Access Control Language

3.1.1 Introduction to Role Based Access Control

In computer systems security, role-based access control (RBAC) (28) is an approach to restricting system access to authorized users. It is a newer alternative approach to mandatory access control (MAC) and discretionary access control (DAC) (29).

The three essential elements in a role-based access control model are user, role and permissions. Roles are associated with sets of permissions, and users are assigned one or more roles to access the authorized resources. Within an organization, roles are relatively stable, while users and permissions are numerous and may change rapidly. Therefore, controlling all access through roles simplifies the management and review of access controls. Figure 7 A RBAC model adopted from (28) shows essential elements in a RBAC model, and the relationship between each other including user-role assignments (UA), role hierarchies (RH), Role Permission Assignments (PA), User-Session Assignment (US), and Role-Session Assignment (RS).

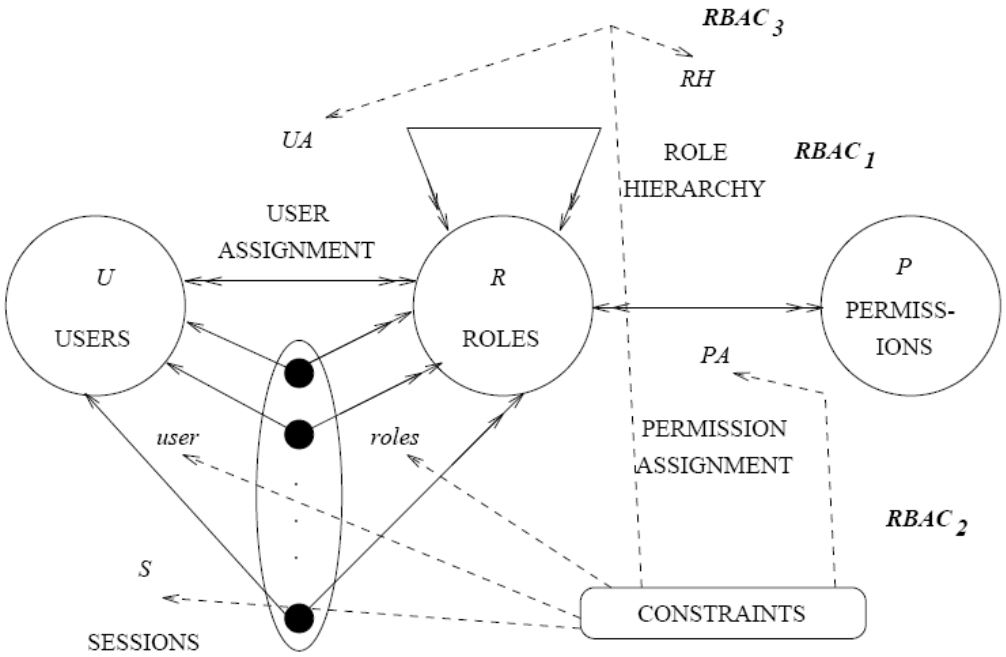

Figure 7 A RBAC model

Since the RBAC model aims at modeling a complex enterprise level access control system, this thesis simplifies the model as an access control CSL (AC_CSL) to fit in the context of services delivery platform of telecom community. Next section introduces the AC_CSL as well as the implementation Padus aspect and an XSLT transformer to transform an AC_CSL to Padus as shown in Figure 8 Access Control CSL to Padus Aspects Transformation.

Figure 8 Access Control CSL to Padus Aspects Transformation

As an experiment to apply rule based access control model to the telecom service delivery platform, AC_CSL provides only the most basic functions. The AC_CSL can expand to provide more function based on more research on the SDP, and the current XSLT transformer can also be easily modified to adapt to provide Padus implementations for extended AC_CSL, since according to the Figure 7 A RBAC model, the more advanced functions largely involves crosscutting concerns, which Padus is intended to address.

3.1.2 Access Control Concern Specific Language

Access Control Concern Specific Language is an XML and Padus based language to address the access control crosscutting concern in the service delivery platform in telecom community. As shown in Figure 9 Access Control CSL model, there are three elements, each role is associated with a set of permissions, and each user is assigned to one or more roles to access the authorized resources.

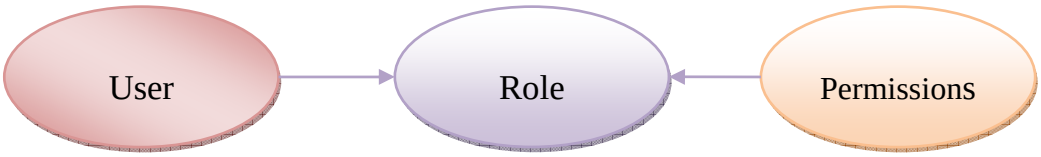

Figure 9 Access Control CSL model

Code 3 Access Control Concern Specific Language shows an example to restrict users can only invoke the services according their role. For example, "AdminUser" can only invoke adminService instead of SMSService or ConfCallService. One user can be assigned with more than one role, such as the "SuperUser", which can invoke all the services in the example, since it's assigned all the three roles.

```
<concern language="AccessControl" name="DSPAccessControl">
  <role name="AdminRole">
    <allow>
      <pointcut name="adminStart(Jp)" pointcut=
"invoking(Jp,'AdminService','AdminPT','creatAdmin')"/>
    </allow>
  </role>
  <role name="SMSRole"> … </role>
  <role name="ConfCallRole"> … </role>
  <user name="AdminUser"> <role name="AdminRole"/> </user>
  <user name="SMSUser"> … </user>
  <user name="ConfCallUser"> … </user>
  <user name="SuperUser">
    <role name="SMSRole"/>
    <role name="AdminRole"/>
    <role name="ConfCallRole"/>
  </user>
</concern>
```

Code 3 Access Control Concern Specific Language

3.2 Padus Implementation

While access control CSL provides a user-friendly interface to capture the access control policies from clients, this policies needs to be transformed into Padus aspects to influence the base BPEL process. Code 4 Padus Implementation for Admin Service Access ControlCode 4 Padus Implementation for Admin Service Access Control shows one of the three aspects generated for the previous AC_CSL example.

We choose to generate one separate aspect for each role rather than one aspect for all the roles to avoid interfering with each other's "currentUser" variables when more than one service is invoked simultaneity. Before invoking an authorized service, Padus aspect requests the role of the current user, and then loops through the all the permissions for the role until it meets required the permit and continue the core process to invoke service, otherwise, an exception will be thrown, and the user requiring service will not be invoked.

```
<aspect>
        <using>
                <namespace name="xmlns:ac" uri="accessControl.example.com"/>
                <partnerLink name="authentication" partnerLinkType="ac:authenticationLT"/>
                <variable name="currentUser" type="ac:user"/>
        </using>
        <pointcut                                                                name="adminStart(Jp)"
pointcut="invoking(Jp,'AdminService','AdminPT','creatAdmin')"/>
        <advice name="accessControl(requiredRole)">
                <sequence>
                        <invoke    partnerLink="authentication"    portType="ac:authenticationPT"
operation="getCurrentUser" outputVariable="currentUser"/>
                        <switch>
                                <case condition="$currentUser=SMSUser">
                                        <switch>
                                                <case condition="$requiredRole=SMSRole">
                                                        <proceed/>
                                                </case>
                                                <otherwise>
                                                        <throw
xmlns:FLT="accessControl.com/faults" faultName="FLT:accessDenied"/>
                                                </otherwise>
                                        </switch>
                                </case>
                                <case condition="$currentUser=AdminUser"> … </case>
                                <case condition="$currentUser=ConfCallUser"> … </case>
                                <case condition="$currentUser=SuperUser"> ... </case>
                                <otherwise>
                                        <throw              xmlns:FLT="accessControl.com/faults"
faultName="FLT:accessDenied"/>
                                </otherwise>
                        </switch>
                </sequence>
        </advice>
        <around joinpoint="Jp" pointcut="adminStart(Jp)">
                <advice name="accessControl(AdminRole)"/>
        </around>
</aspect>
```

Code 4 Padus Implementation for Admin Service Access Control

3.3 XSLT transformation

Extensible Stylesheet Language Transformations (XSLT) (30) is an XML based language to transform XML document from a source tree to a result tree. It functions by associating element patterns in the source tree to templates, and templates generate the result tree according to the information from the matched pattern. While expressing the matching pattern and query the information from the source tree, XML Path language (XPath) (31), whose primary goal is to address parts of XML document.

Code 5 XSLT transform file shows our AC_CSL to Padus transforming XSLT. The first template in code 5, “accessControlAspectTemplate” generates a Padus aspect every time when it matches a pointcut element by both transforming the information it gets from the “pointcut” element and calling other templates, which could perform the same behavior as the first template recursively, until no more action is available according to the XSLT rules.

Until now, AC_CSL and XSLT cooperate with each other to address the procedural level access control crosscutting concerns in a user-friendly way, with Padus as the underlying implementation to affect the base business processes. To further facilitate clients to apply CSL to the services composition, next section introduce a transforming framework, which generates Padus aspect automatically, given the any CSL program and corresponding XSLT transforming rules.

```
<xsl:template name="accessControlAspectTemplate" match="//pointcut">
	<aspect>
		<using>
			<namespace name="xmlns:ac" uri="accessControl.example.com"/>
			<partnerLink name ="authentication"
partnerLinkType="ac:authenticationLT"/>
			<variable name="currentUser" type="ac:user"/>
		</using>
		<xsl:copy-of select="."/>
		<xsl:call-template name="accessControlAdviceDefinitionTemplate"/>
		<around joinpoint="Jp" pointcut="{@name}">
			<advice name="accessControl({ancestor::role[1]/@name})"/>
		</around>
	</aspect>
</xsl:template>
<xsl:template name="accessControlAdviceDefinitionTemplate">
	<advice name="accessControl(requiredRole)">
		<sequence>
			<invoke partnerLink="authentication" portType="ac:authenticationPT"
operation="getCurrentUser" outputVariable="currentUser"/>
			<switch>
				<xsl:for-each select="//user">
					<xsl:call-template name="matchUserTemplate"/>
				</xsl:for-each>
				<otherwise>
					<throw xmlns:FLT="accessControl.com/faults"
faultName="FLT:accessDenied"/>
				</otherwise>
			</switch>
		</sequence>
	</advice>
</xsl:template>
<xsl:template name="matchUserTemplate">
	<case condition="$currentUser={@name}">
		<switch>
			<xsl:for-each select="role">
				<xsl:call-template name="matchUserRoleTemplate"/>
			</xsl:for-each>
			<otherwise>
				<throw xmlns:FLT="accessControl.com/faults"
faultName="FLT:accessDenied"/>
			</otherwise>
		</switch>
	</case>
</xsl:template>
<xsl:template name="matchUserRoleTemplate">
	<case condition="$requiredRole={@name}">
		<proceed/>
	</case>
</xsl:template>
```

Code 5 XSLT transform file

4 CSL to Padus Transforming Framework

Service creation environment described in section 2.5.5 provides a user-friendly service creation environment. Padus described in section 2.5.3 addresses the procedural-level crosscutting concern modularization problem. CSLs introduced in section 2.5.4 facilitates users to deal with crosscutting concerns without the in-depth knowledge of Padus. The essential motivation of CSL is to integrate Padus into the SCE to achieve an even higher level user friendly interface to compose the services.

The motivation of the transforming framework proposed in this chapter is to integrate CSL into the SCE as a plug in on the Eclipse platform. The framework performs two main functions: transforming a given CSL program to Padus aspect for implementing the crosscutting concern, and a syntax validator to check the given CSL program against the XML Schema or DTD to make sure the CSL code is a valid input.

The implementation of the framework involves the Java API for XML Processing (JAXP) (32). JAXP is one of the Java XML programming APIs. It aims at validating and parsing XML documents. The Document Object Model parsing interface (DOM) and the Simple API for XML parsing interface (SAX) are the two major XML parsing interfaces. In our implementation, we use SAX to validate the CSL program, since SAX is faster and uses less memory.

4.1 Transformation

As shown in Figure 10 CSL to Padus Transformer class diagram, for each Concern Specific Language, there should be one specific Transformer class which handles a given kind of CSL program, and generates the Padus aspects into the given output address. Each CSL transformer keeps the information of the XSLT transforming rules specific to the corresponding CSL.

4.2 Validation

Validator is responsible for check if the input CSL program conforms to the predefined syntax. Both DTD and XML Schema can be used to define the syntax of CSLs, as shown in Figure 11 CSL syntax validation class diagram.

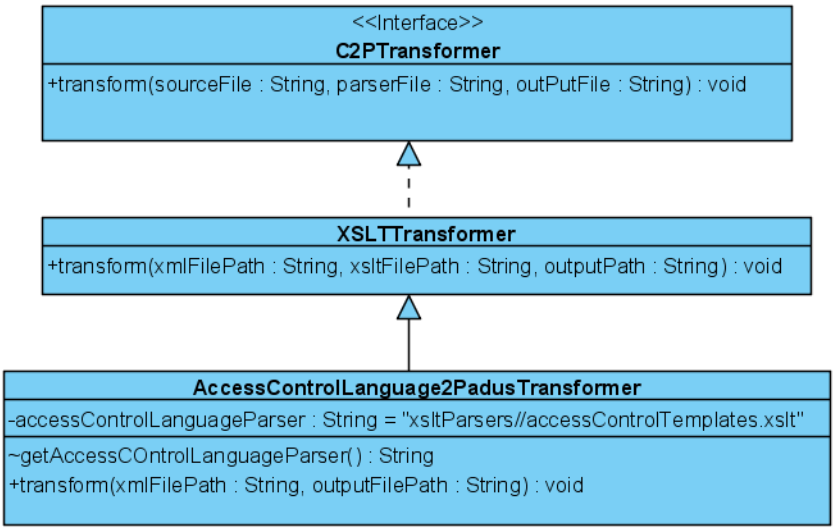

Figure 10 CSL to Padus Transformer class diagram

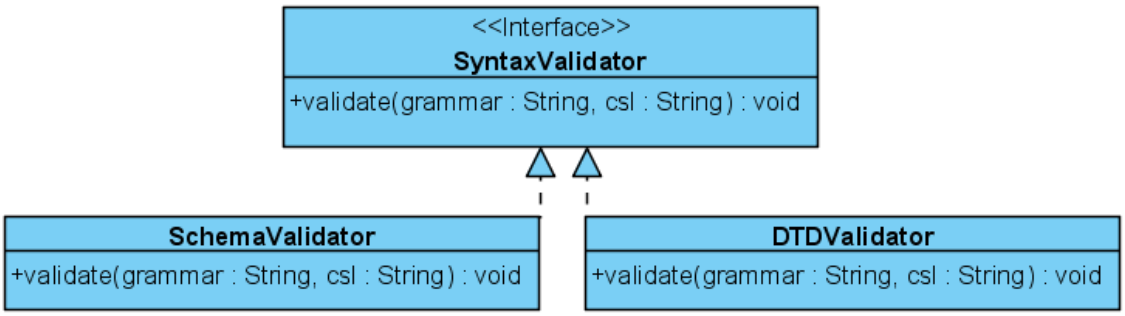

Figure 11 CSL syntax validation class diagram

5 Conclusion

The traditional workflow and business process modeling languages suffer from two problems in providing user an easy to use environment: lack support for visual creation environment and poor modularization of crosscutting concerns. Padus and visual service creation environment addresses these problems by proposing concern specific language for individual crosscutting concern and integrating CSLs into the service creation environment. In this thesis, we propose a role based access control language and its implementation to address the procedural level crosscutting concern in the telecom community. Additionally, we introduce a transformation framework to transform the CSL codes into Padus aspects so as to assist service creation environment.

The role based access control concern specific language applies role based access control notation to address the access control concern in the service delivery platform of telecom community. As an experimental proposal, our model provides only the basic functionality of access control. In future work, a more sophisticated model with support for managing role hierarchies, role permission assignments, user-session assignment, and role-session assignment (28) or other constrains specific to the telecom community can be adopted. The transformation framework also provides only the most basic functionality currently. We believe that JAXP API can fulfill the more sophisticated tasks from the further research on the industry demand.

Bibliography

1. Web Services Architecture. *W3C Working Group Note.* [Online] February 11, 2004. http://www.w3.org/TR/ws-arch/.

2. *Orchestration and Choreography: Standards, Tools and Technologies for Distributed Workflows.* **Ross-Talbot, Steve.** Naples, Italy : s.n., 2005. Workflows management: new abilities for the biological information overflow.

3. **Standard, OASIS.** Web Services Business Process Execution Language Version 2.0. [Online] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, April 11, 2007. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.

4. *Isolating Process-Level Concerns Using Padus.* **Braem, M., Verlaenen, K., Joncheere, N., Vanderperren, W., Van Der Straeten, R., Truyen, E., Joosen, W. and Jonckers, V.** Vienna, Austria : LNCS Springer-Verlag, September 2006. . In Proceedings of the 4th International Conference on Business Process Management (BPM 2006).

5. *AO4BPEL: An Aspect-Oriented Extension to BPEL.* **Anis Charfi, Mira Mezini.** Special issue on "Recent Advances in Web Services", s.l. : World Wide Web Journal (Springer), 2007.

6. *Towards aspect weaving applications.* **Carine Courbis, Anthony Finkelstein.** St. Louis, MO, USA : ACM Press, 2005. International Conference on Software Engineering, Proceedings of the 27th international conference on Software engineering.

7. *Unraveling Crosscutting Concerns in Web Services Middleware.* **Verheecke, B., Vanderperren, W. and Jonckers, V.** s.l. : In IEEE Software journal, pp 42-50, January 2006, Vol. 23(1).

8. **Mathieu Braem, Niels Joncheere, Wim Vanderperren, Ragnhild Van Der Straeten, Viviane Jonckers.** Concern-Specific Languages in a Visual Web Service Creation Environment. *Electronic Notes in Theoretical Computer Science (ENTCS).* 2007, Vol. Volume 163 , Issue 2 .

9. **Hugo Haas, Allen Brown.** Web Services Glossary. *W3C Working Group Note.* [Online] http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/, February 11, 2004. http://www.w3.org/TR/ws-gloss/.

10. **courses, Service Oriented Architecture (SOA).** Web Services Standards. *Technologies and Standards for Service Oriented Architecture Implementation.* [Online] 2005,2006. https://www-304.ibm.com/jct09002c/university/scholars/courseware/repository/SOA/SW719/SW719Topic1.pdf.

11. *Web Services Interoperability Organization.* [Online] http://www.ws-i.org/.

12. **Keith Ballinger, David Ehnebuske, Martin Gudgin, Mark Nottingham, Prasad Yendluri.** Basic Profile Version 1.0. [Online] http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html, 04 16, 2004. http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html.

13. SOAP Version 1.2 Part 0: Primer (Second Edition). *W3C Recommendation.* [Online] http://www.w3.org/TR/2007/REC-soap12-part0-20070427/, April 27, 2007. http://www.w3.org/TR/soap12-part0/.

14. **David Booth, Canyang Kevin Liu.** Web Services Description Language (WSDL) Version 2.0 Part 0: Primer. *W3C Recomendation.* [Online] http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626, June 26 , 2007. http://www.w3.org/TR/wsdl20-primer/.

15. UDDI Specification. *OASIS.* [Online] http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm.

16. **Voormann, H.** Image:Webservices.png. *wikipedia.* [Online] May 18, 2004. http://en.wikipedia.org/wiki/Image:Webservices.png.

17. **Steve Ross-Talbot, Tony Fletcher.** Web Services Choreography Description Language: Primer. *W3C Working Draft.* [Online] http://www.w3.org/TR/2006/WD-ws-cdl-10-primer-20060619/, June 19, 2006. http://www.w3.org/TR/ws-cdl-10-primer/.

18. *Orchestration and Choreography: Standards, Tools and Technologies for Distributed Workflows.* **Ross-Talbot, Stephen.**

19. Business Process Modeling Notation Information. *Object Management Group (OMG).* [Online] 07 09, 2007. http://www.bpmn.org/.

20. Appendix A. AspectJ Quick Reference. *The AspectJTM Programming Guide.* [Online] 2003. http://www.eclipse.org/aspectj/doc/released/progguide/quick.html.

21. *Static and Dynamic Approaches to Weaving.* **Michal Forgac, Jan Kollar.** Poprad, Slovakia : s.n., 2007. 5th Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence and Informatics.

22. **Andrei Popovici, Thomas Gross, and Gustavo Alonso.** *Dynamic Weaving for Aspect-Oriented Progamming.* Notherlands : ACM, 2002.

23. *JAsCo: an aspect-oriented approach tailored for component based software development.* **Davy Suvée, Wim Vanderperren,Viviane Jonckers.** Boston, Massachusetts : ACM Press, 2006. Proceedings of the 2nd international conference on Aspect-oriented software development.

24. *Aspect-oriented logic meta programming.* **De Volder, K.** 1998. Workshop on Aspect Oriented Programming.

25. **De Volder, K,.** *Type-Oriented Logic Meta Programming.* s.l. : Vrije Universiteit Brussel, 1998. Vol. PhD thesis.

26. **Arie van Deursen, Paul Klint, Joost Visser.** Domain Specific Languages: An Annotated Bibliography. [Online] 02 09, 2000. http://homepages.cwi.nl/~arie/papers/dslbib/#foot85.

27. Domain-specific programming language. *wikipedia*. [Online] 08 09, 2007. http://en.wikipedia.org/wiki/Domain-specific_programming_language.

28. *Role Based Access Control Models.* **Ravi S. Sandhu, Edward J. Coynek, Hal L. Feinstein, Charles E. Youman.** s.l. : IEEE Press, 1996, Vols. 29(2): 38-47.

29. *The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments.* **P. A. Loscocco, S. D. Smalley, P. A. Muckelbauer, R. C. Taylor, S. J. Turner, and J. F. Farrell.** Oct. 1998. Proceedings of the 21st National Information Systems Security Conference. Vols. pages 303–314 .

30. **Clark, James.** *XSL Transformations (XSLT) Version 1.0.* [W3C Recommentation] s.l. : W3C, 1999.

31. **James Clark, Steve DeRose.** XML Path Language (XPath) Version 1.0. *W3C recommendation.* [Online] http://www.w3.org/TR/1999/REC-xpath-19991116 , 11 06, 1999. http://www.w3.org/TR/xpath.

32. Java API for XML Processing (JAXP). *Sun Developer Network (SDN).* [Online] 2007. http://java.sun.com/webservices/jaxp/.