EAI Endorsed Transactions
on e-Learning
Research Article ICST.ORG

A Service-oriented Architecture for the Provision and Ranking of Student-Oriented Courses★

Naseem Ibrahim[1,*]

[1]Penn State Behrend, Erie PA, USA

Abstract

Research has long proved that student learn in different styles. Different students have different capabilities and needs. On the other hands, in the last few years the popularity of online degrees has dramatically increased. In current online learning model the school specifies the courses required to obtain a degree. For each course the instructor specifies the course elements including teaching method and assessments. But this contradicts the fact that different students have different capabilities and constraints. Most institutions provide the same courses. A student should be able to select the course that best matches his capabilities and constraints as long as it satisfies the required course outcomes. To achieve this goal, this paper proposes the use of Service-oriented Architecture (SOA). This paper introduces an extended service-oriented architecture and an extended service definition, which will enable the specification and provision of student-oriented courses. This paper also proposes a student-oriented course composition approach and a student-oriented course ranking approach.

Received on 30 September, 2015; accepted on 29 February, 2016; published on 10 March, 2016

Keywords: Student-oriented, SOA, Context, Service Model

doi:10.4108/eai.10-3-2016.151119

1. Introduction

Research in education [7][25] has long suggested that students learn through different learning styles. These styles include sequential, verbal, visual, active, reflective, sensing and intuitive. It has long been proven that different students have different capabilities and needs.

In the last few years, the high cost of education and the increased number of adult students resulted in the wide popularity of online degrees.

In the current education model, a school defines the classes a student needs to complete to obtain a degree. For each course, the instructor defines the elements of the teaching process with respect to instruction method, assessment types and schedule. But this contradicts the basic fact that different students have different capabilities and needs. A student should be able to choose a course from any institution. The course that best matches its capabilities and constrains. The only constraint is that the course should satisfy the required outcomes. We call this model student-oriented learning.

To achieve the student-oriented learning model, this research suggests the uses of *Service-Oriented Computing (SOC)* [20]. SOC is a computing paradigm that uses *service* as the fundamental element for application development processes. An architectural model of SOC in which *service* is a first class element is called *Service-Oriented Architecture (SOA)* [14]. We believe a course can be represented as a service and can be provided by a service-oriented architecture.

Current service-oriented architectures in its current state are not sufficient for achieving this goal. They focus on functionality during service provision. Hence, in Section 2, we propose an *Extended Service-oriented Architecture (ESOA)* that supports the provision of student-oriented courses.

In the newly introduced ESOA, a course is specified as a service. Current services model functionality and some nonfunctional properties. But that is not enough. A richer service that is able to represent a student-oriented course is needed. Hence, in Section 3, we extend the definition of a service by including the concept *Context* to support the rich definition of courses. We represent student capabilities and constraints using "Context". Context has been defined [11] as the information used to characterize the situation of an

★Please ensure that you use the most up to date class file, available from EAI at http://doc.eai.eu/publications/transactions/latex/

*Corresponding author. Email: nii1@psu.edu

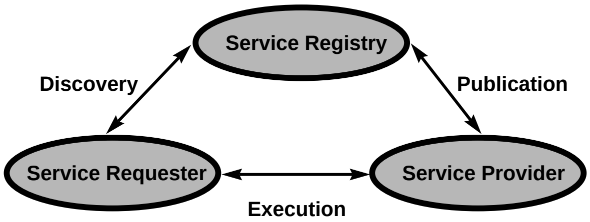

Figure 1. Traditional SOA Architecture

entity. This entity can be a person, a place, or an object. The context representation and the logic of context proposed by Wan [47] for reasoning about context-awareness are suitable formalisms for enriching SOA modeling. The extended service is defined formally to support formal verification.

In ESOA, a student should be able to specify all his requirements, capabilities and constrains in a rich definition. Hence, in Section 4 we introduce *Student-oriented Course Requirement Definition (SOCRD)* language. SOCRD will be used by course requesters to specify the students requirements.

For a student to complete a degree, he should complete the degree required outcomes by completing multiple courses. This is basically a composition of courses. Hence, in Section 5 we introduce a course composition approach that composes courses defined using our extended service. The composition is formally defined to support formal verification.

In many cases multiple courses can partially meet the requirements of the course requester. In such cases a ranking is necessary. Hence, Section 6 introduces a ranking approach that ranks courses while considering the requirements, capabilities and constraints of a course requester.

Section 7 presents a brief study of related work. Section 8 provides an extended example. Finally, Section 9 presents some concluding remarks and future work.

2. Extended SOA

Figure 1 illustrates the elements of a traditional Service-oritened Architecture (SOA). It consists of three main modules, the service provider, the service requester and the service registry. The service provider publishes a service definition in the service registry. The service requester searches the service registry and selects from the published services. After selecting a service, the service requester interacts with the service provider by sending requests and receiving responses.

In traditional SOA, the publication, discovery and execution of services are heavily based on the functionality of the services. The service provider publishes the functionality of the service in the registry. The service requester searches the registry looking for services that matches its requirements in terms of functionalities. But such architectures are not sufficient for the publication of our student-oriented courses. Hence, this section introduces an *Extended SOA (ESOA)* that

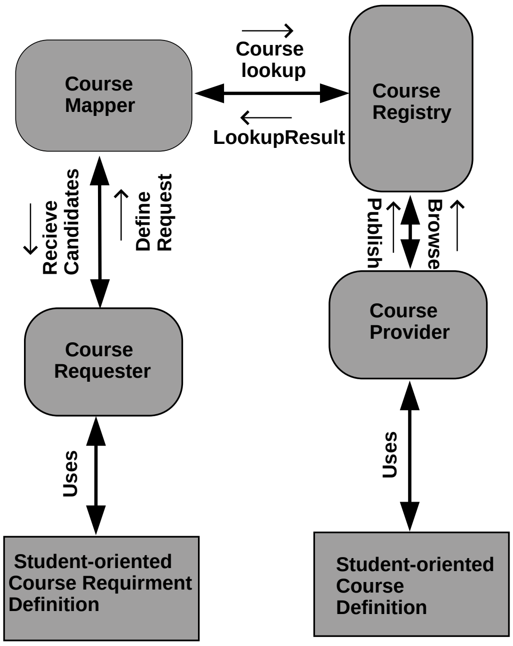

Figure 2. Extended SOA Architecture

supports the publication, discovery and execution of student-oriented courses.

ESOA enables course providers to define rich courses for the provision of student-oriented courses. It also enables course requesters to obtain courses that best match their requirements while considering their capabilities and constraints. Figure 2 illustrates the architecture of ESOA which consists of the following elements:

- **Course Requester**: It is the entity that is requiring a course. It represents the client side of the interaction. It is usually a student who is looking for a course that best meets his requirements while respecting the student's capabilities and constraints.

- **Course Provider**: It is the entity that provides a course. Course providers publish course descriptions on registries to enable automated discovery and invocation.

- **Student-oriented Course Requirement Definition**: In ESOA, a course requester is able to specify and list all the course requirements, his capabilities and constrains in a rich definition. This entity enables this rich definition.

- **Student-oriented Course Definition** : To enable the best possible matching and discovery of courses, course providers has to publish a rich definition of a course. Traditional definition of services that relay on service functionality is not sufficient. Hence, a rich course definition is required. This entity is responsible for achieving this.

- **Course Registry**: This entity is responsible for enabling the discovery of student-oriented courses. Course providers publish their rich course definition in the course registry. The course mapper searches the course registry looking for courses that matches the course requester requirements while respecting the course requester capabilities and constraints.
- **Course Mapper:** This unit is responsible for the following three main responsibilities.
  1. It is responsible for matching the requirements of the course requester to the available course in the course registry. The novelty in the matching process is that is does not only focus on the functional requirements, it takes into consideration the capabilities and constraints of the course requester.
  2. It is responsible for ranking available courses is case of multiple matches. The ranking process considers the requirements of the course requester.
  3. It is responsible for the composition of courses in case no single course is sufficient to satisfy the requirements of the course requester.

3. Student-oriented Course Definition

As in any business interaction, a course provider has the main goal of reaching the largest possible number of consumers. A consumer for a course is the course requester or student. To enable the discovery of the courses a provider can deliver, the course provider publishes course information in a course registry. This paper suggested the use of services.

In a traditional SOA, the service definition includes the service functionality. But, in a student-oriented environment, the publication of the course functionality is not enough. Hence, traditional services are not sufficient for the specification and publication of course information. To reach the largest possible number of requester, a course provider has to define his courses in a richer definition.

To enable such rich definitions, this paper extended the definition of a service by including the concept *Context*. Context is used to represent student capabilities and constraints. Context has been defined [11] as the information used to characterize the situation of an entity. This entity can be a person, a place, or an object. The context representation and the logic of context proposed by Wan [47] for reasoning about context-awareness are suitable formalisms for enriching SOA modeling. Beside context, this paper adds nonfunctional properties and legal rules to the definition of a function. All these elements are incapsulated in what is called *ExtendedService*.

The rest of this section introduces the informal and formal definition of *ExtendedService*. *ExtendedService* has been formally defined to enable the formal composition and verification of the composition results.

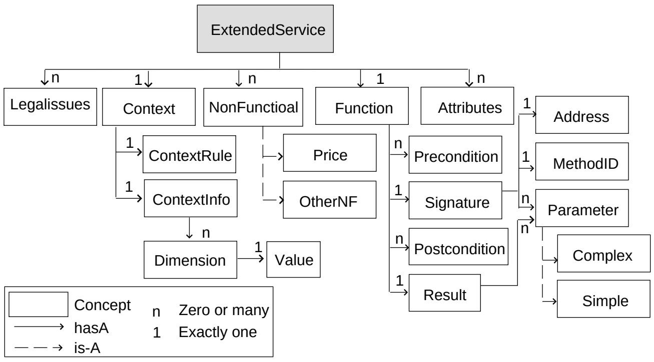

Figure 3. *ExtendedService* Structure

3.1. An Informal Definition of ExtendedService

An *ExtendedService* is divided into the following parts, as shown in Figure 3.

1. *Functionality*: Its definition includes the function *signature*, *result*, *precondition* and *postcondition*. The *signature* part defines the function *identifier*, the invocation *address*, and the *parameters* of the function. The function invocation has the same effect as in a programming environment, since service function is an autonomous program. Each parameter has an *identifier* and a *type*. The *result* part defines the returned data of the service function. The *precondition* should be made true, either by the service provider or the consumer, in order to make the function available. The *postcondition* is guaranteed by the service provider to be true after service execution.

2. *Nonfunctional properties*: The nonfunctional properties associated with the service are listed in this section. Pricing information, which can itself be a complex property expressing different prices for different amount of buying, is an example of nonfunctional property.

3. *Attributes*: Every attribute is a type-value pair. Attributes provide sufficient information that is unique to a service. As an example, for providing a course the appropriate attributes may include title of course and institution name.

4. *Legal issues:* Business rules and trade laws that are enforced at the locations of service provision and service delivery are included in this section. Example policies govern *refund*, *administrative charges*, *penalties*, and *service requesters rights*. Such rules are expressible as logical expressions in predicate logic.

5. *Context:* The context part of the contract is divided into *context info* and *context rules*. The contextual information of the service provider is specified in

Functionality	**Name: ProgrammingII** **Precondition: ProgrammingI == true** **Postcondition: ProgrammingII == true**
Attributes	**Title: ProgrammingII** **Description: This course continues the study of basic programming concepts in Java.** **Institute: USA University**
Nonfunctional	**Price: = 750$**
Legal Issues	**Refund Condition: 100% refund if withdrawn in less than 5 days from start.** **Payment methods: Credit cards only** **Payment schedule: Payment should be received by 5th day of classes.** **Discounts: Students with GPA more 3.5 gets 20% discount.**
Context	**Context Info: [Location : New York] ^ [Duration:4 weeks] ^ [AssessmentType: Exams & Assignments] ^ [AssessmentLocation: Online] ^ [TeachingMethod: Lectures] ^ [Attendence: Optional]** **Context Rule: student-city in USA ^ age > 18**

Figure 4. ProgrammingII ExtendedService

the *context info* section. The situation or context rule that should be true for service delivery is specified in *context rules* section. It is the responsibility of the service requester to validate the *context info* for obtaining the service, and it is the responsibility of the service provider to validate the *context rules* at service delivery time.

Example 1. Figure 4 illustrates an example of a service that was modeled using the novel ExtendedService definition. The service is for a Programming II course that is being provided by USA University.

3.2. Formal Definition of ExtendedService

An *ExtendedService* is formally defined using a model-based specification notation. The context information encapsulated in the ExtendedService is written in the notation introduced by Wan [47]. Because of this underlying formalism it is possible to rigorously verify the claims made in an *ExtendedService*. Below we briefly give the formal representation of the *ExtendedService* elements.

Let $\mathbb{C}$ denote the set of all such logical expressions. $X \in \mathbb{C}$ is a constraint. The following notation is used in our definition:

- $\mathbb{T}$ denotes the set of all data types, including abstract data types.
- $Dt \in \mathbb{T}$ means Dt is a datatype.
- $v : Dt$ denotes that v is either constant or variable of type Dt.
- X_v is a constraint on v. If v is a constant then X_v is true.
- V_q denotes the set of values of data type q.
- $x :: \Delta$ denotes a logical expression $x \in \mathbb{C}$ defined over the set of parameters Δ. A parameter is a 3-tuple, defining a data type, a variable of that type, and a constraint on the values assumed by the variable. We denote the set of data parameters as $\Lambda = \{\lambda = (Dt, v, X_v) | Dt \in \mathbb{T}, v : Dt, X_v \in \mathbb{C}\}$.

1. Functionality: An *ExtendedService* provides a single function. This functionality is defined to include the function *signature*, *result* value, *preconditions* and *postconditions*.

Definition 1. A service function is a 4-tuple $f = \langle g, i, pr, po \rangle$, where g is the function signature, i is the function result, pr is the precondition, and po is the post-condition. A signature is a 3-tuple $g = \langle n, d, u \rangle$, where $n : string$ is the function identification name, $d = \{x | x \in \Lambda\}$ is the set of function parameters and $u : string$ is the function address, the physical address on a network that can be used to call a function. For example, it can be an IP address. The result is defined as $i = \langle m, q \rangle$, where $m : string$ is the result identification name and $q = \{x | x \in \Lambda\}$ is the set of parameters resulting from executing the *ExtendedService*. The precondition pr and postcondition po are data constraints. That is, $pr :: z, z \subseteq \Lambda$ and $po :: z, z \subseteq \Lambda$

2. Nonfunctional properties: Typical nonfunctional properties associated with the service are pricing and maintenance information. Pricing can be formalized as follows.

Definition 2. Nonfunctional property list is $\kappa = \langle p, \ldots \rangle$, where p is the service cost and ... denote other nonfunctional properties. The service cost p is defined as a 3-tuple $p = \langle a, cu, un \rangle$, where $a : \mathbb{N}$ is the price amount defined as a natural number, $cu : cType$ is currency tied to a currency type $cType$, and $un : uType$ is the unit for which pricing is valid. As an example, $p = (100, \$, hour)$ denotes the pricing of 100$/hour. Other nonfunctional properties can be similarly defined using appropriate data types and included in κ.

3. Attributes: These include some semantic information that is unique to a service.

Definition 3. An attribute has a name and type, and is used to define some semantic information associated with the service. As an example, each *ExtendedService* can be given a unique identifier, a version number, and type of release. They are defined as service attributes. The set of attributes is $\alpha = \{(Dt, v_\alpha) | Dt \in \mathbb{T}, v_\alpha : Dt\}$.

4. Legal issues: As part of the contract in an *ExtendedService*, a set of legal rules that constrain the contract may be included.

Definition 4. A legal issue is a rule, expressed as a logical expression in $\mathbb{C}$. A rule may imply another, however no two rules can conflict. We write $l = \{y | y \in \mathbb{C}\}$ to represent the set of legal rules.

5. Context: Both *context information* and *context rules* are formally specified in a contract. These two parts provide context-awareness ability to *ExtendedService*.

Context information is formally specified, as defined in [47], using *dimensions* and *tags* along the dimensions.

In our research, we have been using the five dimensions *WHERE*, *WHEN*. *WHAT*, *WHO*, and *WHY*. In general, it is the responsibility of service providers to choose as many dimensions and their names in order to present the contexts associated with services. Assume that the service provider has invented a finite set $DIM = \{X_1, X_2, \ldots, X_n\}$ of dimensions, and associated with each dimension X_i a type τ_i. Following the formal aspects of context developed by Wan [47], we define a context c as an aggregation of ordered pairs (X_j, v_j), where $X_j \in DIM$, and $v_j \in \tau_j$.

A *context rule* is a situation which might be true in some contexts and false in some others. For example, the situation $VERYWARM = Temp > 40 \wedge Humid > 70$, is true only in contexts where the temperature is greater than 40 degrees, and the humidity is greater than 70.

Definition 5. A context is formalized as a 2-tuple $\beta = \langle r, c\rangle$, where $r \in \mathbb{C}$, built over the contextual information c. Context information is formalized using the notation in [47]: Let $\tau : DIM \rightarrow I$, where $DIM = \{X_1, X_2, \ldots, X_n\}$ is a finite set of dimensions and $I = \{a_1, a_2, \ldots, a_n\}$ is a set of types. The function τ associates a dimension to a type. Let $\tau(X_i) = a_i$, $a_i \in I$. We write c as an aggregation of ordered pairs (X_j, v_j), where $X_j \in DIM$, and $v_j \in \tau(X_j)$.

Putting these definitions together we arrive at a formal definition for *ExtendedService*.

Definition 6. A *ExtendedService* is a 5-tuple $s = \langle f, \kappa, \alpha, l, \beta\rangle$, where f is the service function, κ is the set of nonfunctional properties, α is the set of service attributes, l is the set of legal rules and β is the context.

Example 2. Example 1 illustrated an informal representation of a simple ExtendedService. Below is the formal representation of the same ExtendedService. Let p denote the *ExtendedService* for providing Programming II course who provides the services described in Figure 4. The formal notation of the *ExtendedService* p is $s_p = \langle f_p, \kappa_p, \alpha_p, l_p, \beta_p\rangle$, where the tuple components are explained below.

1. Function: $f_p = \langle g_p, i_p, pr_p, po_p\rangle$ where,
   - Function signature: $g_p = \langle n_p, d_p, u_p\rangle$, where $n_p = (ProgrammingII)$ is the name, $d_p = \{(Location, string), (age, int), (CourseList, string[])\}$ are input data parameters, and $u_p = (XXX)$ is the address.
   - Function result: $i_p = \langle m_p, q_p\rangle$, where $m_p = (ResultP)$ is the name and the set of output data parameters is $q_p = \{(PassedCourse, bool), (Balance, double)\}$.
   - Function precondition: $pr_p = (ProgrammingI == true)$.
   - Function postcondition $po_p = (ProgrammingII == true)$.
2. Nonfunctional: $\kappa_p = \langle p_p\rangle$, $p_p = \langle a_p, cu_p, un_p\rangle$, where $a_p = (750)$ is the cost, $cu_p = (dollar)$ is the currency, and $un_p = (course)$ is the pricing unit.
3. Attributes: $\alpha_p = \{(title = ProgrammingII), (Description = Thiscourse...), (institute = USAUniversity)\}$.
4. Legal: $l_p = \{(RefundFull \ if \ DropDate - StartDate < 5Days), (PaymentMethod == Credit), (PaymentDate <= start + 5, (Discount\ 20\% \ if \ GPA > 3.5)\}$.
5. Context: $\beta_p = \langle r_p, c_p\rangle$, where $r_p = \{(studentCity \ in \ USA), (age > 18)\}$ is the context rule and $c_p = \{(Location = NewYork), (Duration = 4weeks), (AssessmentType = exams\&Assignments), (AssessmentLocation = Online), (TeachingMethod = Lectures), (Attendance = Optional)\}$ is the contextual information.

4. Student-oriented Course Requirement Definition

In a student-oriented learning model, the student who is a course requester is aiming to get a course that best matches his requirements and needs. In ESOA, the requirements and constraints defined by the course requester are passed to the Course Mapping Unit. A traditional service query using a traditional query language that focuses on functionality is not sufficient to specify the requester requirements and constraints.

Hence, this section introduces *Student-oriented Course Requirement Definition (SOCRD)* language. SOCRD enables service providers to specify the required functionality, non-functional requirements, constraints defined using context, and required legal rules.

Below is an informal and formal definition of a request defined using SOCRD. The formalism of SOCRD is necessary to enable the formal verification of the matching result generated by the Mapping Unit.

4.1. Informal Definition of SOCRD

Figure 5 shows the structure of a course request defined using SOCRD. Each course request will consist of the four parts *required function*, *required legal issues*, *required nonfunctional properties*, and *requester and consumer context*. The course requester is responsible for defining these requirements.

The course requester can also assign a weight to each requirement. This weight defines the priority of each requirement and is used in ranking the set of candidate course when the Course Mapping unit performs matching.

- *Required Function*: The required functional properties defines the functionality required by the course

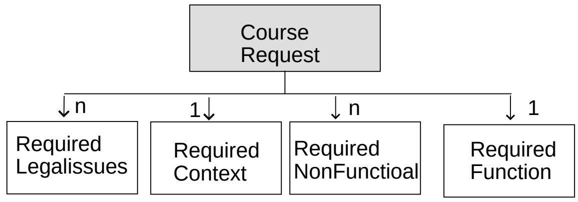

Figure 5. Course Request Structure

requester and is defined in terms of the functionality preconditions and postconditions. For each element the course requester can assign a weight.

- *Required Nonfunctional Properties*: The required nonfunctional properties defines the nonfunctional properties required by the course requester. The definition of the nonfunctional properties in SOCRD is identical to the definition of the nonfunctional properties in ExtendedService. The only exception is the addition of the weights.

- *Required Legal Issues*: This section contains the required legal rules specified by the course requester. Its definition is also identical to the definition in ExtendedService with the addition of the weights.

- *Required Context*: This section includes the contextual information of the course requester and provider. It also uses the same definition of the contextual information in ExtendedService. A weight value can also be added to each requirement.

4.2. Formal Definition of SOCRD

The formal definition of the legal rules requirements is identical to Definition 4. The formal definition of the required context is identical to Definition 5.

The functional requirements is defined using preconditions and postconditions. It is defined as follows:

Definition 7. The required function is defined as $\hat{f} = \langle \hat{pr}, \hat{po} \rangle$, where $\hat{pr}$ is the set of preconditions of the required function, and $\hat{po}$ is the set of postconditions of the required function. The formal definitions of precondition and postcondition are identical to the one given in Definition 1.

The nonfunctional requirements lists the maximum acceptable price. It is defined as follows:

Definition 8. The required nonfunctional property is defined as $\hat{\kappa} = \langle \hat{p} \rangle$, where $\hat{p}$ is the maximum price required. The formal definition of required price is identical to the definition given in Definition 2.

Putting the definition of all the elements of a course requirements and adding weight to it, will give us the formal definition below.

Definition 9. A course request r_e is defined as $r_e = \langle \hat{f}, \hat{\kappa}, \hat{\beta}, \hat{l}, \Xi \rangle$, where $\hat{f}$ is a course required function, $\hat{\kappa}$ is the nonfunctional requirement, $\hat{l}$ is the legal rules requirements, $\hat{\beta}$ is the required contextual information of the course requester, and $\Xi : (x \in \{Low, BelowAverage, Average, AboveAverage, High, Exact\}) \rightarrow (y \in \{\hat{pr}, \hat{po}, \hat{p}, \hat{l}, \hat{\beta}\})$ is a function that assign weights to the elements of the course request.

5. Course Composition

In ESOA, a course requester passes his requirements and constrains defied using SOCRD to the Mapping Unit. The Mapping Unit is responsible for matching the requester request with *ExtendedServices* published in the Course Registry. In some cases, a single course cannot satisfy the requirements of the requester. In such cases, the Mapping unit is responsible of composing two or more courses to satisfy the requester requirements.

The rest of this section discusses the composition of courses defined as *ExtendedServices*.

A review of available composition approaches, discussed in the Related Works Section, shows that most available service composition approaches are not formal. There are a few exceptions, however these formal approaches focus only on composing service functionalities. Hence, the are not sufficient for the composition of courses defined as *ExtendedServices* and there is a need for a new formal composition method.

The composition method presented in this section is both formal and complete. Formal composition constructs and their semantics are defined. It is complete in the sense that the composition is defined on all parts of *ExtendedService*, not just on service functionality. The primary advantage of formalism and completeness is that complex expressions of composed *ExtendedServices* can be constructed and subjected to formal analysis. Formal analysis is necessary because service expressions are often complex, involving many composition operators, and hard to do by manual inspection at execution time.

The Course Mapping unit creates a *course expression* involving the names of *ExtendedServices* and composition constructs. All composition constructs in a course expression have the same precedence, and hence a course expression is evaluated from left to right. To enforce a particular order of evaluation, parenthesis may be used. The result of evaluating a course expression is an *ExtendedService*.

In the context of courses, two types of compositions are necessary sequential and parallel. In sequential compositions one course is a prerequisite for another course. In a parallel composition two courses can be completed concurrently. The following are the composition constructs for representing sequential and parallel composition.

- Sequential Construct $\gg$: Given two *ExtendedServices* A and B, the service expression $A \gg B$ defines an *ExtendedService* C which is the sequential composition of A and B. The intended execution behavior of the *ExtendedService* C is the execution behavior of B after the execution of A.

- Parallel Construct $\|$: Given two *ExtendedServices* A and B, the course expression $A\|B$ defines the parallel composition of A and B. The parallel composition $A\|B$ models the concurrent executions of *ExtendedServices* A and B. Therefore the resulting behavior of this composite service should be the merging of their individual behaviors in time order.

Below we let $A = \langle f_A, \kappa_A, \alpha_A, l_A, \beta_A\rangle$, and $B = \langle f_B, \kappa_B, \alpha_B, l_B, \beta_B\rangle$ denote two *ExtendedServices*, where $f_A = \langle g_A, i_A, pr_A, po_A\rangle$, $f_B = \langle g_B, i_B, pr_B, po_B\rangle$, $g_A = \langle n_A, d_A, u_A\rangle$, $g_B = \langle n_B, d_B, u_B\rangle$, $i_A = \langle m_A, q_A\rangle$, $i_B = \langle m_B, q_B\rangle$, $\kappa_A = \langle p_A\rangle$, $\kappa_B = \langle p_B\rangle$, $\beta_A = \langle r_A, c_A\rangle$, and $\beta_B = \langle r_B, c_B\rangle$. The result of a composition is an *ExtendedService*.

5.1. Sequential Composition

The sequential composition $A \gg B$ of *ExtendedServices* A and B is an *ExtendedService*, expressed as the tuple $\langle f_{A\gg B}, \kappa_{A\gg B}, \alpha_{A\gg B}, l_{A\gg B}, \beta_{A\gg B}\rangle$ whose components are defined below.

Function. : $f_{A\gg B} = \langle g_{A\gg B}, i_{A\gg B}, pr_{A\gg B}, po_{A\gg B}\rangle$, $g_{A\gg B} = \langle n_{A\gg B}, d_{A\gg B}, u_{A\gg B}\rangle$, $i_{A\gg B} = \langle m_{A\gg B}, q_{A\gg B}\rangle$, where

$g_{A\gg B}$:

$n_{A\gg B}$	=	$n_A \frown n_B$	naming convention
$d_{A\gg B}$	=	$d_A \cup d_B$	combine input data parameters
$u_{A\gg B}$	=	$\{u_A, u_B\}$	both addresses are necessary

$i_{A\gg B}$:

$m_{A\gg B}$	=	$m_A \frown m_B$	naming convention
$q_{A\gg B}$	=	$q_A \cup q_B$	combine output parameters
$pr_{A\gg B}$	=	$pr_A \cup (pr_B \setminus po_A)$	if B requires more constraints
$pr_{A\gg B}$	=	pr_A	if B doesn't require more constraints
$po_{A\gg B}$	=	$po_A \cup po_B$	if po_A is not used as an input of B
$po_{A\gg B}$	=	po_B	if po_A is absorbed as an input for B

Nonfunctional Properties. : $\kappa_{A\gg B} = \langle p_{A\gg B}\rangle$ where, $p_{A\gg B} = \langle a_{A\gg B}, cu_{A\gg B}, un_{A\gg B}\rangle$ where $cu_{A\gg B} = cu_A = cu_B$, $un_{A\gg B} = un_A = un_B$, and

$$a_{A\gg B} = \begin{cases} a_A + a_B & \text{normal pricing} \\ max\{a_A, a_B\} & \text{promotional} \\ min\{a_A, a_B\} & \text{special sale} \end{cases}$$

Attributes. : $\alpha_{A\gg B} = \alpha_A \cup \alpha_B$

Legal Issues. : $l_{A\gg B} = l_A \cup l_B$, defined as the union of the issues of A and B.

Context. : We use the semantics of context union ($\sqcup$) and sub-context ($\sqsubset$), as defined by Wan [47]. These are defined essentially using relational semantics. For *ExtendedService* A the context is $\beta_A = \langle r_A, c_A\rangle$. This means that r_A is true in context c_A in order that A may be provided. Once the service A has been provided, the context and rules that are true in that context should be computed. Letting these rules r'_A and the context c'_A, we need to merge them with r_B and c_B, $\beta_B = \langle r_B, c_B\rangle$ to arrive at $\beta_{A\gg B}$. With this rationale, we define $\beta_{A\gg B} = \langle r_{A\gg B}, c_{A\gg B}\rangle$, $r_{A\gg B} = r'_A \cup r_B$, and $c_{A\gg B} = c'_A \sqcup c_B$, the smallest closure of contexts c'_A and c_B. It is expected that $c'_A \sqsubset c_B$ holds for most of the applications, because anything outside of c_B can be ignored.

5.2. Parallel Composition

The parallel composition $A\|B$ of the *ExtendedServices* A and B is a *ExtendedService*, expressed as the tuple $\langle f_{A\|B}, \kappa_{A\|B}, \alpha_{A\|B}, l_{A\|B}, \beta_{A\|B}\rangle$ whose components are defined below.

Function. : $f_{A\|B} = \langle g_{A\|B}, i_{A\|B}, pr_{A\|B}, po_{A\|B}\rangle$, $g_{A\|B} = \langle n_{A\|B}, d_{A\|B}, u_{A\|B}\rangle$, $i_{A\|B} = \langle m_{A\|B}, q_{A\|B}\rangle$, where

$g_{A\|B}$:

$n_{A\|B}$	=	$n_A \frown n_B$	naming convention
$d_{A\|B}$	=	$d_A \cup d_B$	combine input data parameters
$u_{A\|B}$	=	$\{u_A, u_B\}$	both addresses are necessary

$i_{A\|B}$:

$m_{A\|B}$	=	$m_A n \frown m_B$	naming convention
$q_{A\|B}$	=	$q_A \cup q_B$	combine output parameters
$pr_{A\|B}$	=	$pr_A \cup pr_B$	preconditions are mutually disjoint
$po_{A\|B}$	=	$po_A \cup po_B$	both postcondition sets are available

Nonfunctional Properties. : $\kappa_{A\|B} = \langle p_{A\|B}\rangle$, where $p_{A\|B} = \langle a_{A\|B}, cu_{A\|B}, un_{A\|B}\rangle$ and:

$$a_{A\|B} = a_A + a_B$$
$$cu_{A\|B} = cu_A = cu_B$$
$$un_{A\|B} = un_A = un_B$$

Attributes. : $\alpha_{A\|B} = \alpha_A \cup \alpha_B$

Legal Issues. : $l_{A\|B} = l_A \cup l_B$, defined as the union of the issues of A and B.

Context. : $\beta_{A\|B} = \langle r_{A\|B}, c_{A\|B}\rangle$, where $r_{A\|B} = r_A \cup r_B$, and $c_{A\|B} = c_A \sqcup c_B$.

6. Student-oriented Course Ranking

In ESOA, the Course Mapper takes course requests from the course requester and searches the course registry for course that matches the requester requirements. In many cases, multiple course might be available and in other cases no exact match is available. The mapper is responsible of ranking the candidate courses. This section discusses the ranking algorithm. The ranking process can be defined in the following 3 steps.

6.1. Form Weight Vector

In formulating a course request the requester assigns a weight to each property that is relevant for him. The mapper extracts these weights and constructs the weight vector, as in Equation 1, where R_w is the weight vector and w_i is the weight of property i as defined by the service requester. Property i can be a precondition, a postcondition, a nonfunctional requirement or a legal requirement. The number of properties n depends on the request defined by the course requester.

$$R_w = [w_1, w_2, w_3, .., w_n] \tag{1}$$

A weight can be *{Low, BelowAverage, Average, AboveAverage, High, Exact}*. An *ExtendedService* that do not satisfy *Exact* values are filtered when doing the matching. So the possible weight values are *{Low, BelowAverage, Average, AboveAverage, High}*. We assume in further discussion that weight values are whole numbers in the range $1 \ldots 5$, where 1 denotes *Low* and 5 denotes *High*.

6.2. Construct Weight Matrix

By using the weight vectors constructed in Step 1, the weight matrix for the candidate courses is constructed. This is shown in Equation 2, where n is the number of properties defined in Equation 1 and m is the number of the candidate courses. Each column represents the weights of the properties in a single course. Each row represents the weights of a single property in the different courses.

$$C_w = \begin{bmatrix} w_{1,1} & w_{2,1} & .. & w_{m,1} \\ w_{1,2} & w_{2,2} & .. & w_{m,2} \\ .. & .. & .. & .. \\ w_{1,n} & w_{2,n} & .. & w_{m,n} \end{bmatrix} \tag{2}$$

The value of the course property weight depends on the property type. If a property j is a precondition, postcondition, or a legal rule (without values), a weight $w_{i,j}$ is equal to 1, if service i satisfies property j and is equal to 0 otherwise. If property j is price, or legal rule (with values), $w_{i,j}$ is calculated according to Equation 3, where z is the required property value as defined in the service request and x is the actual property value specified in the candidate courses.

$$w_{i,j} = \begin{cases} 1 & \text{if } x \le z \\ 1 - (\frac{x-z}{2z-z}) = 2 - \frac{x}{z} & \text{if } z < x < 2z \\ 0 & \text{if } x \ge 2z \end{cases} \tag{3}$$

Equation 3 assumes that actual value that is more than double the required value will be given a weight of 0. Anything that is less than the required value will be given 1. And an actual value between the required value and double the required value will be given a weight that depends on how close the actual value is to the required value. For example, if the required price as defined in the course request is 500, an actual course price of 550 should be given a better weight than a price of 800.

6.3. Calculate Weights for Ranking

A single weight value for each candidate course is computed, and the courses are ranked based on these weights. Equation 4 uses the results of steps one and two to calculate the raking weight vector.

$$W = R_w \times C_w \tag{4}$$

The ranking weight vector W contains the weights of the different candidate courses. These weights are used to rank the courses. The candidate course with the highest weight value is placed first in the candidate course list. The course with the second highest weight value is placed second in the candidate course list and so on for the rest of the candidate courses.

Example 3. A course requester is looking for an Intermediate French course. The location of the course requester is in New York City. The course requester is requiring that the price be 500$ with a weight *Average*. The course requester is requiring that the course provider should be 200 miles away, with a weight *High*.

Two courses IntermediateFrenchA and IntermediateFrenchB provide the functionality required in by the course requester. They don't provide an exact match to the price and location, but rather a partial match. The list of properties will include: $\{RequiredPrice, RequiredDistance\}$. Hence, the request weight vector is

$$R_w = \begin{bmatrix} Average & High \end{bmatrix}$$

In numbers,

$$R_w = \begin{bmatrix} 3 & 5 \end{bmatrix}$$

Course IntermediateFrenchA (rsA) has a cost of $rsA_c =$ 400\$ and is 300 miles away $rsA_d = 300miles$. Course IntermediateFrenchB (rsB) has a cost of $rsB_c = 600$\$ and is 200 miles away $rsB_d = 200miles$. Hence, the course weight matrix is defined, using Equations 2 and 3, as:

$$C_w = \begin{bmatrix} w_{rsA,c} & w_{rsB,c} \\ w_{rsA,d} & w_{rsB,d} \end{bmatrix}$$

where, $w_{rsA,c} = 1$, $w_{rsB,d} = 1$ and:

$$w_{rsA,d} = 2 - \frac{600}{500} = 0.8$$

$$w_{rsB,c} = 2 - \frac{300}{200} = 0.5$$

The ranking weight vector will then be defined using Equation 4 as:

$$W = \begin{bmatrix} 3 & 5 \end{bmatrix} \begin{bmatrix} 1 & 0.5 \\ 0.8 & 1 \end{bmatrix} = \begin{bmatrix} 5.4 & 6.5 \end{bmatrix}$$

Hence, course IntermediateFrenchA scores 5.4 and ranked second, while course IntermediateFrenchB scores 6.5 and ranked first. Although the first course is more expensive, the ranking reflected that fact that the course requester is more concerned with the course provider location.

7. Related Work

The related work can be classified into three main areas: related service models, related service composition approaches and related service provision approaches.

7.1. Related Service Models

The modeling approaches can be classified based either on the language, or the architecture or a combination of both. The two main languages that have been used for modeling services are UML [1, 32], and WSDL with the related Web description languages [24, 31, 38, 51]. Architecture based service modeling approach uses an Architectural Definition Language (ADL) [10, 27] to describe services. There are a few other methods [6, 16] which combine language and some abstract architectural details for describing service features.

The UML-based language UML4SOA [48] supports a model-driven development of SOA architecture. No precise guidelines exist for creating such an architecture. This approach relies mainly on the intuition of the developer, and lacks formalism. The family of Web Services Description Languages (WSDL) and OWL-S (including SWS) [29, 31, 51]) have been used to model services. Semantic embedding of data is enabled by SWS, however these languages are not formal. They do not provide any support for stating legal rules, and offer no verification support.

The three architectural description languages SOADL [10, 27], SRML [16], and SOFM [6] provide formal notations for modeling services. But they have no support for the context.

Analysis of related service models shows that while some approaches are formal and others have limited support for context, no single approach is formal and include context as a first class element.

7.2. Related Composition Approaches

For the sake of placing our work in the right place among others, we have chosen to discuss two types of service composition approaches pursued in the literature. These are (1) Web services based approaches, and (2) formal approaches.

The two main Web Services approaches for syntactic service composition are *orchestration* and *choreography* [43]. BPEL [9] is the most important orchestration approach while WS-CDL [49] is an example of choreography approach. The main difference between BPEL and WS-CDL is that WS-CDL describes a global view of the observable behavior of message exchanges of the participating service, while BPEL describes the behavior from the point of view of the orchestrator [43]. Neither approach is formal or consider context in the composition process.

From the formal side, we choose *Automata*, *Petri nets*, and *Process Algebras* approaches for comparison. We restrict to a discussion on how compositions are done, assuming their formal notations.

Many authors [33], [17], [30], [18], [13], and [12] have used automata to model services and their compositions. One group has used BPEL and another group has used WS-CDL. The basic idea is to use a two-step transformation. In the first step the BPEL or WS-CDL model is transformed to an automaton. In the second step either UPPAAL [3] or SPIN [4] model checker is used for model checking.

From the many published studies [35][34][26][39][23], that use Petri nets we have chosen two categories of work to review. One approach is to transform language models to Petri nets, and the second approach is to enhance Petri nets directly for service compositions. Although, both categories are formal they lack the support for context.

Approaches that uses process algebra such as Calculus for Orchestration of Web Services (COWS) [44], The Service Centered Calculus (SCC) [5], and the Service Oriented Computing Kernel (SOCK) [22], are formal but they also lack the support for context composition.

7.3. Related Service Provision Approaches

The most notable related service provision approaches are SeGSeC [19], eFlow [8], SELF-SERV [40], SHOP2 [50], SWORD [37], Argos [2], FUSION [45], Proteus [21], SPACE [28], StarWSCoP [42], METEOR-S [46], SeCSE [36], DynamiCoS [41] and TSCN [15]. Most of these approaches does not consider context in the provision of services. The few that do provide no formal definition of context and does not consider the relationship between context and functionality. Hence, the formal verification of compositions is not possible and they cannot be used in the specification of our rich courses.

8. Example

Example 1 introduced an informal representation of an ExtendedService for a ProgrammingII course that is being taught by USA University. Example 2 presented a formal representation of the same ExtendedService. This section extends the previous example by introducing two new ExtendedServices. The first new ExtendedService provides a ProgrammingII course that is being taught by UK University. The second new ExtendedService provides a DataStructure course that is being taught by USA University. The informal representation of the new ExtendedServices are presented in Figures 6 and 7.

Functionality	Name: ProgrammingII Precondition: ProgrammingI == true Postcondition: ProgrammingII == true
Attributes	Title: ProgrammingII Description: This course continues the study of basic programming concepts in Java. Institute: UK University
Nonfunctional	Price: = 600$
Legal Issues	Refund Condition: 80% refund if withdrawn in less than 5 days from start. Payment methods: Credit cards only Payment schedule: Payment should be received by 5th day of classes. Discounts: Students with GPA more 3.5 gets 20% discount.
Context	Context Info: [Location : London ^ [Duration:5 weeks] ^ [AssessmentType: Exams & Assignments] ^ [AssessmentLocation: Online] ^ [TeachingMethod: Lectures] ^ [Attendence: Optional]

Figure 6. New ProgrammingII ExtendedService

Functionality	Name: DataStructure Precondition: ProgrammingII == true Postcondition: DataStructure == true
Attributes	Title: Data Strucure Description: This course introduces the concepts of data structure using Java. Institute: USA University
Nonfunctional	Price: = 550$
Legal Issues	Refund Condition: 100% refund if withdrawn in less than 5 days from start. Payment methods: Credit cards only Payment schedule: Payment should be received by 5th day of classes. Discounts: Students with GPA more 3.0 gets 15% discount.
Context	Context Info: [Location : New York] ^ [Duration:4 weeks] ^ [AssessmentType: Exams & Assignments] ^ [AssessmentLocation: Online] ^ [TeachingMethod: Lectures] ^ [Attendence: Optional] Context Rule: student-city in USA ^ age > 18

Figure 7. DataStructure ExtendedService

The formal representation of the DataStructure ExtendedService is presented below:

Let ds denote the *ExtendedService* for providing DataStructure course who provides the services described in Figure 4. The formal notation of the *ExtendedService* ds is $s_{ds} = \langle f_{ds}, \kappa_{ds}, \alpha_{ds}, l_{ds}, \beta_{ds} \rangle$, where the tuple components are explained below.

1. Function: $f_{ds} = \langle g_{ds}, i_{ds}, pr_{ds}, po_{ds} \rangle$ where,
   - Function signature: $g_{ds} = \langle n_{ds}, d_{ds}, u_{ds} \rangle$, where $n_{ds} = (DataStructure)$ is the name, $d_{ds} = \{(Location, string), (age, int), (CourseList, string[])\}$ are input data parameters, and $u_{ds} = (YYY)$ is the address.
   - Function result: $i_{ds} = \langle m_{ds}, q_{ds} \rangle$, where $m_{ds} = (ResultDS)$ is the name and the set of output data parameters is $q_{ds} = \{(PassedCourse, bool), (Balance, double)\}$.
   - Function precondition: $pr_{ds} = (Programming|I == true)$.
   - Function postcondition $po_{ds} = (DataStructure == true)$.
2. Nonfunctional: $\kappa_{ds} = \langle p_{ds} \rangle$, $p_{ds} = \langle a_{ds}, cu_{ds}, un_{ds} \rangle$, where $a_{ds} = (550)$ is the cost, $cu_{ds} = (dollar)$ is the currency, and $un_{ds} = (course)$ is the pricing unit.
3. Attributes: $\alpha_{ds} = \{(title = DataStructures), (Description = Thiscourse...), (institute = UKUniversity)\}$.
4. Legal: $l_{ds} = \{(RefundFull\ if\ DropDate - StartDate < 5Days), (PaymentMethod == Credit), (PaymentDate <= start + 5, (Discount\ 15\%\ if\ GPA > 3.0)\}$.
5. Context: $\beta_{ds} = \langle r_{ds}, c_{ds} \rangle$, where $r_{ds} = \{(studentCity\ in\ USA), (age > 18)\}$ is the context rule and $c_{ds} = \{(Location = NewYork), (Duration = 5weeks), (AssessmentType = exams\&Assignments), (AssessmentLocation = Online), (TeachingMethod = Lectures), (Attendance = Optional)\}$ is the contextual information.

To illustrate the formal composition of ExtendedServices, below is the sequential composition of the ExtendedService presented in Example 2 and the DataStructure ExtendedService:

Let $p \gg ds$ denote the *ExtendedService* for providing Programming II and DataStructure courses. The formal notation of the *ExtendedService* $p \gg ds$ is $s_{p \gg ds} = \langle f_{p \gg ds}, \kappa_{p \gg ds}, \alpha_{p \gg ds}, l_{p \gg ds}, \beta_{p \gg ds} \rangle$, where the tuple components are explained below.

1. Function: $f_{p \gg ds} = \langle g_{p \gg ds}, i_{p \gg ds}, pr_{p \gg ds}, po_{p \gg ds} \rangle$ where,
   - Function signature: $g_{p \gg ds} = \langle n_{p \gg ds}, d_{p \gg ds}, u_{p \gg ds} \rangle$, where $n_{p \gg ds} = (Programming|| + DataStructure)$ is the name, $d_{p \gg ds} = \{(Location, string), (age, int), (CourseList, string[])\}$ are input data parameters, and $u_{p \gg ds} = (YYY)$ is the address.
   - Function result: $i_{p \gg ds} = \langle m_{p \gg ds}, q_{p \gg ds} \rangle$, where $m_{p \gg ds} = (ResultPDS)$ is the name and the set of output data parameters is $q_{p \gg ds} = \{(PassedCourse, bool), (Balance, double)\}$.

Required Functionality	**Precondition: ProgrammingI == true** **Priority: Exact** **Postcondition: ProgrammingII == true** **Priority: Exact**
Required Nonfunctional	**Price: = 750$** **Priority: High**
Required Legal Issues	**Refund Condition: 100% refund** **Priority: Low**
Required Context	**Context Info: [Location : New York]** **Priority: Average**

Figure 8. Informal Course Request

- Function precondition: $pr_{p\gg ds} = (ProgrammingI == true \wedge ProgrammingII == true)$.
- Function postcondition $po_{p\gg ds} = (ProgrammingII == true \wedge DataStructure == true)$.

2. Nonfunctional: $\kappa_{p\gg ds} = \langle p_{p\gg ds}\rangle$, $p_{p\gg ds} = \langle a_{p\gg ds}, cu_{p\gg ds}, un_{p\gg ds}\rangle$, where $a_{p\gg ds} = (550 + 750 = 1300)$ is the cost, $cu_{p\gg ds} = (dollar)$ is the currency, and $un_{p\gg ds} = (course)$ is the pricing unit.

3. Attributes: $\alpha_{p\gg ds} = \{(title = ProgrammingII + DataStructures), (Description = Thiscourse..., andThiscourse...), (institute = USAUniversity)\}$.

4. Legal: $l_{p\gg ds} = \{(RefundFull\ \ if\ \ DropDate - StartDate < 5Days), (PaymentMethod == Credit), (PaymentDate <= start + 5, (Discount\ 15\%\ if\ GPA > 3.0))\}$.

5. Context: $\beta_{p\gg ds} = \langle r_{p\gg ds}, c_{p\gg ds}\rangle$, where $r_{p\gg ds} = \{(studentCity\ in\ USA), (age > 18)\}$ is the context rule and $c_{p\gg ds} = \{(Location = NewYork), (Duration = 4 + 5 = 9weeks), (AssessmentType = exams\&Assignments), (AssessmentLocation = Online), (TeachingMethod = Lectures), (Attendance = Optional)\}$ is the contextual information.

A user is looking for a ProgrammingII course, his requirements are listed in Figure 8. His requirements can be formally defined using SOCRD as follows:

Let r_p denote the ProgrammingII course request. The request is formally defined as $r_p = \langle \hat{f_p}, \hat{\kappa_p}, \hat{\beta_p}, \hat{l_p}, \Xi_p\rangle$, where:

- $\hat{f_p}$=$\{\hat{pr}, \hat{po}\rangle$ where $\hat{pr} = \langle (ProgrammingI == true)\rangle$ and $\hat{po} = \langle (DataStructure == true)\}$.
- $\hat{\kappa_p}$=$\{(700, \$, course)\}$.
- $\hat{l_p} = \{(Refund == 100\%)\}$.
- $\hat{\beta_p} = \{(Location == NewYork)\}$.
- $\Xi_p = \{((a == 700), High), ((ProgrammingII == true), Exact), ((DataStructure == true), Exact), ((Refund == 100), Low), ((Location == NewYork), Average)\}$

Two ExtendedServices provided ProgrammingII, the ExtendedService presented in 1 and the ExtendedService provided in this section. We will call these ExtendedServices ES1 and ES2 respectfully, Because two ExtendedServices provide the same course a ranking is necessary. Below is the ranking of these two ExtendedServices according to the user requirements.

EX1 and EX2 provide an exact match to the required pre and post conditions. They don't provide exact match to price, discount and location, but rather a partial match. The list o properties will include: $\{RequiredPrice, RequiredRefund, RequiredLocation\}$. Hence, the request weight vector is

$$R_w = \begin{bmatrix} High & Low & Average \end{bmatrix}$$

In numbers,

$$R_w = \begin{bmatrix} 5 & 1 & 3 \end{bmatrix}$$

Course EX1 $(EX1)$ has a cost of $EX1_c = 750\$$,it is located in New York, and has a refund amount of 100% $EX1_r = 20$. Course EX2 $(EX2)$ has a cost of $EX2_c = 600\$$, it is located in London, and has a refund amount of 80% $EX2_r = 80$. Hence, the course weight matrix is defined, using Equations 2 and 3, as:

$$C_w = \begin{bmatrix} w_{EX1,c} & w_{EX2,c} \\ w_{EX1,l} & w_{EX2,l} \\ w_{EX1,r} & w_{EX2,r} \end{bmatrix}$$

where, $w_{EX2,c} = 1$, $w_{EX1,l} = 1$, $w_{EX2,l} = 0$, $w_{EX1,r} = 1$ and:

$$w_{EX1,c} = 2 - \frac{750}{700} = 0.93$$

$$w_{EX2,r} = 2 - \frac{100}{80} = 0.75$$

The ranking weight vector will then be defined using Equation 4 as:

$$W = \begin{bmatrix} 5 & 1 & 3 \end{bmatrix} \begin{bmatrix} 0.93 & 1 \\ 1 & 0 \\ 1 & 0.75 \end{bmatrix} = \begin{bmatrix} 8.65 & 7.25 \end{bmatrix}$$

Hence, course EX1 scores 8.65 and ranked first, while course EX2 scores 7.25 and ranked first.

9. Conclusion and Future Work

It has long been proven that different students have different capabilities and needs. Being able to adhere to the needs of all students, might not be possible in traditional face-to-face classes. But when it comes to online education, achieving this might be easier. To enable students to select courses that best meets their requirements, the following should be achieved:

- A course provider should be able to publish a rich definition of courses.
- A course requester should be able to define a rich request of his requirements.
- A student-oriented framework should match the students requirements with available courses.

The work presented in this paper utilizes SOA to achieve the above goals. Courses are defined using context-aware services, while student requests are defined using context-aware queries. A context-aware framework is responsible for the publication, discovery and provision of the student-oriented courses.

In addition this paper has presented an extended service-oriented architecture, a formal extended service model, a formal composition theory, and a student-oriented ranking approach.

We are currently working on a complete implementation of the newly introduced architecture and associated tools.

References

[1] (2008) Service oriented architecture modeling language (SOAML) - specification for the UML profile and metamodel for services (UPMS), OMG Submission document: ad/2008-11-01. Available at http://www.omgwiki.org/SoaML/doku.php?id=specification.

[2] Ambite, J.L. and Weathers, M. (2005) Automatic composition of aggregation workflows for transportation modeling. In *Proceedings of the 2005 national conference on Digital government research* (Digital Government Society of North America): 41–49.

[3] Behrmann, G., David, A. and Larsen, K.G. (2004) A tutorial on UPPAAL. In *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004* (Springer–Verlag), **LNCS 3185**: 200–236.

[4] Ben-Ari, M. (2008) *Principles of the Spin Model Checker* (Springer).

[5] Boreale, M., Bruni, R., Nicola, R.D., Lanese, I., Loreti, M., Montanari, U., Sangiorgi, D. *et al.* (2006) Scc: a service centered calculus. In *Proceedings of WS-FM 2006, 3rd International Workshop on Web Services and Formal Methods, Lecture Notes in Computer Science* (Springer): 38–57.

[6] Cao, X.X., Miao, H.K. and Xu, Q.G. (2008) Modeling and refining the service-oriented requirement. In *TASE '08: Proceedings of the 2008 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering* (Washington, DC, USA: IEEE Computer Society): 159–165.

[7] Carver, C.A., J., Howard, R. and Lane, W. (1999) Enhancing student learning through hypermedia courseware and incorporation of student learning styles. *Education, IEEE Transactions on* **42**(1): 33–38. doi:10.1109/13.746332.

[8] Casati, F., Ilnicki, S., Jin, L.J., Krishnamoorthy, V. and Shan, M.C. (2000) Adaptive and dynamic service composition in eflow. In *Proceedings of the 12th Int'l Conference on Advanced Info. Systems Engineering* (Springer-Verlag): 13–31.

[9] Curbera, F., Khalaf, R., Mukhi, N., Tai, S. and Weerawarana, S. (2003) The next step in web services. *Commun. ACM* **46**(10): pp. 29–34. doi:http://0-doi.acm.org.mercury.concordia.ca/10.1145/944217.944234.

[10] Dan, X., Shi, Y., Tao, Z., Xiang-Yang, J., Zao-Qing, L. and Jun-Feng, Y. (2006) An approach for describing soa. In *International Conference on Wireless Communications, Networking and Mobile Computing, WiCOM 2006*: 1–4.

[11] Dey, A.K. (2001) Understanding and using context. *Personal Ubiquitous Comput.* **5**(1): 4–7. doi:http://dx.doi.org/10.1007/s007790170019.

[12] Dong, J.S., Liu, Y., Sun, J. and Zhang, X. (2006) Verification of computation orchestration via timed automata. In *ICFEM06* (Springer–Verlag), **LNCS 4260**: 226–245.

[13] Dyaz, G., Cambronero, M.E., Pardo, J.J., Valero, V. and Cuartero, F. (2006) Automatic generation of correct web services choreographies and orchestrations with model checking techniques. In *International Conference on Internet and Web Applications and Services/Advanced International Conference on Telecommunications, 2006. AICT-ICIW '06.*: 186 – 186. doi:10.1109/AICT-ICIW.2006.53.

[14] Erl, T. (2007) SOA *Principles of Service Design* (Upper Saddle River, NJ, USA: Prentice Hall PTR).

[15] Fan, G., Yu, H., Chen, L. and Liu, D. (2009) An approach to analyzing dynamic trustworthy service composition. In Gómez-Pérez, A., Yu, Y. and Ding, Y. [eds.] *The Semantic Web, Fourth Asian Conference, ASWC 2009, Shanghai, China, December 6-9, 2009. Proceedings* (Springer), *Lecture Notes in Computer Science* **5926**: 261–275.

[16] Fiadeiro, J.L., Lopes, A. and Bocchi, L. (2006) A formal approach to service component architecture. In Bravetti, M., Núñez, M. and Zavattaro, G. [eds.] *Web Services and Formal Methods. LNCS, vol 4184* (Springer, Berlin Heidelberg), 193–ï£¡213.

[17] Foster, H., Uchitel, S., Magee, J. and Kramer, J. (2003) Model-based verification of web service compositions. In *Proc. of the eighteen* IEEE *international conference on automated software engineerting* ASE03: 152–163.

[18] Fu, X., Bultan, T. and Su, J. (2004) Analysis of interacting bpel web services. In *Proceedings of the 13th international conference on World Wide Web (WWW '04)* (New York, NY, USA: ACM): 621–630. doi:http://doi.acm.org/10.1145/988672.988756.

[19] Fujii, K. and Suda, T. (2009) Semantics-based context-aware dynamic service composition. *ACM Trans. on Autonomous and Adaptive Systems* **4**(2): 1–31. doi:http://doi.acm.org/10.1145/1516533.1516536.

[20] Georgakopoulos, D. and Papazoglou, M.P. (2008) *Service-Oriented Computing* (The MIT Press).

[21] Ghandeharizadeh, S., Knoblock, C., Papadopoulos, C., Shahabi, C., Alwagait, E., Ambite, J.L., Cai, M. *et al.* (2003) Proteus: A system for dynamically composing and intelligently executing web services. In *Proceedings of the 1st International Conference on Web Services* (Las Vegas, NV, USA).

[22] Guidi, C., Lucchi, R., Gorrieri, R., Busi, N. and Zavattaro, G. (2006) Sock: a calculus for service oriented computing. In *Proceedings of the 4th International Conference on Service-Oriented Computing, volume 4294 of LNCS* (Chicago, IL, USA: Springer): 327–338.

[23] Hamadi, R. and Benatallah, B. (2003) A petri net-based model for web service composition. In *Proceedings of the 14th Australasian database conference* (Darlinghurst, Australia: Australian Computer Society, Inc.): 191–200.

[24] Herrmann, M., Aslam, M.A. and Dalferth, O. (2007) Applying semantics (wsdl, wsdl-s, owl) in service oriented architectures (soa). In *Proceedings of the 10th Intl. Protege Conference* (Budapest, Hungary).

[25] Heywood, J. (2005) *Learning Strategies and Learning Styles* (Wiley-IEEE Press), 119–151. doi:10.1109/9780471744696.ch5, URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5271343.

[26] Hinz, S., Schmidt, K. and Stahl, C. (2005) Transforming bpel to petri nets. In *Proceedings of the International Conference on Business Process Management (BPM2005), volume 3649 of Lecture Notes in Computer Science* (Springer-Verlag): 220–235.

[27] Jia, X., Ying, S., Zhang, T., Cao, H. and Xie, D. (2007) A new architecture description language for service-oriented architecture. In *Sixth International Conference on Grid and Cooperative Computing (GCC 2007)*: 96 –103.

[28] Jin, C., Wu, M. and Ying, J. (2009) A structure-based approach for dynamic services composition. *Journal of Software* **4**(8): 891–898.

[29] Kashyap, V., Bussler, C. and Moran, M. (2008) *The Semantic Web, Semantics for Data and Services on the Web* (Springer).

[30] Kazhamiakin, R., Pandya, P. and Pistore, M. (2006) Timed modelling and analysis in web service compositions. In *The First International Conference on Availability, Reliability and Security (ARES 2006)*: 7. doi:10.1109/ARES.2006.134.

[31] Martin, D., Paolucci, M., McIlraith, S. and et al, M. (2004) Bringing semantics to web services: The owl-s approach. In *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)* (San Diego, California, USA).

[32] Mayer, P., Schroeder, A. and Koch, N. (2008) Mdd4soa: Model-driven service orchestration. In *EDOC '08: Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference* (Washington, DC, USA: IEEE Computer Society): 203–212. doi:http://dx.doi.org/10.1109/EDOC.2008.55.

[33] Mitra, S., Kumar, R. and Basu, S. (2007) Automated choreographer synthesis for web services composition using i/o automata. In *IEEE International Conference on Web Services (ICWS 2007)*: 364 –371. doi:10.1109/ICWS.2007.47.

[34] Narayanan, S. and McIlraith, S.A. (2002) Simulation, verification and automated composition of web services. In *Proceedings of the 11th international conference on World Wide Web* (New York, NY, USA: ACM): 77–88. doi:http://0-doi.acm.org.mercury.concordia.ca/10.1145/511446.511457.

[35] Ouyang, C., Verbeek, E., van der Aalst, W.M.P., Breutel, S., Dumas, M. and ter Hofstede, A.H.M. (2007) Formal semantics and analysis of control flow in ws-bpel. *Science of Computer Programming* **67**(2-3): 162–198. doi:http://0-dx.doi.org.mercury.concordia.ca/10.1016/j.scico.2007.03.002.

[36] Penta, M.D., Bastida, L., Sillitti, A., Baresi, L., Maiden, N., Melideo, M., Tilly, M. *et al.* (2009) Secse–service centric system engineering: An overview. In Nitto, E.D., Sassen, A.M., Traverso, P. and Zwegers, A. [eds.] *At Your Service: Service-Oriented Computing from an EU Perspective* (The MIT Press), 241–272.

[37] Ponnekanti, S.R. and Fox, A. (2002) Sword: A developer toolkit for web service composition. In *Proceedings of the 11th International WWW Conference*.

[38] Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A. *et al.* (2005) Web service modeling ontology. *Applied Ontology* **1**(1): 77–106.

[39] Rosario, S., Benveniste, A., Haar, S. and Jard, C. (2006) Foundations for web services orchestrations: Functional and *QoS* aspects, jointly. In *Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2006)*: 309 –316. doi:10.1109/ISoLA.2006.8.

[40] Sheng, Q.Z., Benatallah, B., Dumas, M. and Mak, E.O.Y. (2002) Self-serv: a platform for rapid composition of web services in a peer-to-peer environment. In *Proceedings of the 28th international conference on Very Large Data Bases* (VLDB Endowment): 1051–1054.

[41] Silva, E., Pires, L.F. and van Sinderen, M. (2009) Supporting dynamic service composition at runtime based on end-user requirements. In *Proceedings of the 1st Workshop on User-generated Services (UGS2009) at the 7th International Joint Conference on Service Oriented Computing, (ICSOC 2009)* (Stockholm, Sweden).

[42] Sun, H., Wang, X., Zhou, B. and Zou1, P. (2003) Research and implementation of dynamic web services composition. In Zhou, X., Jähnichen, S., Xu, M. and Cao, J. [eds.] *Advanced Parallel Processing Technologies, 5th InternationalWorkshop, APPT 2003* (Springer-Verlag), *Lecture Notes in Computer Science* **2834**: 457–466.

[43] ter Beek, M.H., Bucchiarone, A. and Gnesi, S. (2007) Formal methods for service composition. *Annals of Mathematics, Computing and Teleinformatics* **1**(5): 1–5.

[44] Tiezzi, F. (2009) *Specification and Analysis of Service-Oriented Applications*. Phd thesis, Università degli Studi di Firenze, Florence, Italy.

[45] VanderMeer, D., Datta, A., Dutta, K., Thomas, H., Ramamritham, K. and Navathe, S.B. (2003) Fusion: A system allowing dynamic web service composition and automatic execution. In *Proceedings of the IEEE Int. Conference on E-Commerce Technology* (IEEE Computer Society): 399.

[46] Verma, K., Gomadam, K., Sheth, A.P., Miller, J.A. and Wu, Z. (2005) *The* METEOR-S *Approach for Configuring and Executing Dynamic Web Processes*. Technical report, LSDIS Lab, University of Georgia, Athens, Georgia.

[47] Wan, K. (2006) *Lucx: Lucid Enriched with Context*. Phd thesis, Concordia University, Montreal, Canada.

[48] Wirsing, M., Bocchi, L., Fiadeiro, J.L., Gilmore, S., Hoelzl, M., Koch, N., Mayer, P. *et al.* (2008) Sensoria: Engineering for Service-Oriented Overlay Computers. In di Nitto, E., Sassen, A.M., Traverso, P. and Zwegers, A. [eds.] *At Your Service: Service Engineering in the Information Society Technologies Program* (MIT Press).

[49] WS-CDL (2005) *Web Services Choreography Description Language Version 1.0*. Tech. rep.

[50] WU, D., PARSIA, B., SIRIN, E., HENDLER, J., NAU, D. and NAU, D. (2003) Automating daml-s web services composition using shop2. In *Proceedings of 2nd International Semantic Web Conference*.

[51] ZAREMBA, M., KERRIGAN, M., MOCAN, A. and MORAN, M. (2006) Web services modeling ontology. In CARDOSO, J. and SHETH, A.P. [eds.] *Semantic Web Services, Processes and Applications* (Springer), 63–87.