Leveraging Web-Services and Peer-to-Peer Networks

Mike P. Papazoglou[1], Bernd J. Krämer[2], and Jian Yang[1]

[1] INFOLAB — Tilburg University, PO Box 90153,
NL-5000 LE Tilburg, The Netherlands
{mikep,jian}@uvt.nl,
[2] FernUniversität Hagen,
D-58084 Hagen, Germany
bernd.kraemer@fernuni-hagen.de

Abstract. Peer-oriented computing is an attempt to weave interconnected machines into the fabric of the Internet. Service-oriented computing (exemplified by web-services), on the other hand, is an attempt to provide a loosely coupled paradigm for distributed processing. In this paper we present an event-notification based architecture and formal framework towards unifying these two computing paradigms to provide essential functions required for automating e-business applications and facilitating service publication, discovery and exchange.

1 Introduction

A large number of enterprises nowadays is implementing a SOAP/WSDL/UDDI layer on top of existing applications or components and is assembling applications by consuming web-services. The manifestation of web-services for such applications is through widely accepted industry standards such as XML, SOAP, WSDL (Web-Services Definition Language) and UDDI (Universal Description, Discovery and Integration protocol). Interactions of web-services occur as SOAP calls carrying XML data content and the service definitions of the web-services are expressed using WSDL as the common (XML-based) standard. WSDL is used to publish a web service in terms of its *ports* (addresses implementing this service), *port types* (the abstract definition of operations and exchanges of messages), and *bindings* (the concrete definition of which packaging and transportation protocols such as SOAP are used to inter-connect two conversing end points). The UDDI standard is a directory service that contains service publications and enables web-service clients to locate candidate services and discover their details.

The characterization of the web-service operation is the classic client/server model. The service provider (server) will register with the UDDI registry and the requester (client) will contact the registry to discover the server location so that it can interact with it. This is a straightforward approach to distributed computing that provides the advantage that clients are coupled to the servers only via a contract mechanism. Since this contract is fully described by using WSDL, developers can construct clients using the contract information. All providers must make their services available by publishing their contract and advertising their service.

Peer-to-Peer (P2P) computing is the sharing of computer resources and services through direct communication between systems. Each functional unit in the network, called a peer, is behaviourally similar and is logically capable of both providing and consuming information. True P2P networks are vastly distributed and do not require a centralized directory for indexing purposes. When a peer decides that data hosted on another peer is useful, it visits directly this peer in order to obtain that data. The P2P network is usually fluctuating and dynamic with peer neighbour relationships breaking and reforming as the load or infrastructure stability changes.

When comparing P2P networks with web-services functionality, we observe that peer-to-peer systems also leverage a service-oriented architecture but have their own idiosyncrasies. Unlike web-services, the determination of who is a provider, a requester or a registrar (of a resource) is much looser. Typically, a peer is all three of the aforementioned roles. However, like web-services, peers must also publish a resource (allowing it to be found and accessed by other peers) with efficient precision for the other peers to be able to broadcast their needs and receive meaningful responses. Publication and discovery are paramount for both paradigms, however, the two approaches diverge on lookup services. Peers use decentralized discovery while web-services use larger centralized directories such as UDDI. Lastly, another similarity is that both web-services and P2P networks have heavy emphasis on distributed computing and on using XML as a means to describe information.

Fortunately, the standards and frameworks used to create web-services can also be utilized to develop P2P applications. This is because both sets of architectures fundamentally coordinate interactions between loosely coupled systems. Utilizing a common framework based on current web-service technologies would enable P2P developers with elementary building blocks for building applications. In fact, JXTA, the P2P framework initiated by Sun Microsystems [10], i s making adjustments to its core platform to make peers interoperate with web-services using protocols like SOAP and WSDL.

This paper examines key intersect points that enable web-services and P2P networks to work together and in particular, looks at ways in which web-services discovery can benefit from P2P decentralization. Our contribution concentrates on an architectural approach and formal framework towards unifying web-services and P2P networks to provide essential functions required for automating e-business applications and facilitating service publication, discovery and exchange.

2 Problems with Web-Service Directories

To open new markets and find new sources of supply enterprises use a common service registry (UDDI) for identifying potential trading partners and for cataloguing their business functions and characteristics. UDDI specification provides two main types of interfaces (APIs): one for describing services and registering service entries in the directory and one for enquiring about service entries and provider characteristics. This allows the services to be dynamically discovered and composed into more complex (value-added) services.

It is expected that vertical sectors will have a variety of specialised UDDI directories that serve their community as a whole offering business functionality on the Web. In fact some *vertical* e-marketplaces, such as semiconductors, travel industry, and automotive industry already provide to their members a unified view of sets of UDDI-based products and services to enable them to transact business using diverse mechanisms, such as web-services. And this is already happening to a large extend. With these vertical e-marketplaces services can be published and hosted throughout the e-marketplace network and used on demand. The goal of web-services when used within the context of e-marketplaces is to enable business solutions by composing and programming web-services, e.g., using the Business Process Execution Language for Web Services (BPEL4WS) [3]. This allows companies to conduct electronic business, by invoking web-services, with all partners in a marketplace rather than with just the ones with whom they have collaborative business agreements. Service offers are described in such a way, e.g., WSDL over UDDI, that they allow automated discovery to take place and offer request matching on functional and non-functional service capabilities.

One of the major problems with the centralized indexing scheme provided by UDDI is that it does not scale well because the number and physical distribution of the UDDI clients can quickly overwhelm this centralized configuration and can lead to serious performance bottlenecks. Adding more servers or implementing load-balancing strategies does not constitute a practical solution as they may prove to be costly and disruptive. In contrast to UDDI, P2P networks content is normally described and indexed locally to each peer and search queries are propagated across the network. In this model no central index is required to span the network.

One of the major points of intersection between P2P and web-service technologies involves bringing the decentralization aspect of P2P networks to the central service discovery mechanisms of web-services provided by UDDI. It is not difficult to envision a P2P network architecture that promotes a logically decentralized arrangement of registered service descriptions and that also provides web-service descriptions much in the same way that UDDI does.

3 A Federated Architecture for P2P Web-Services

To enable the fusion of web services and P2P computing we can employ a federation of UDDI-enabled peer registries that operate in a decentralized fashio rather than requiring each peer to publish their own service descriptors locally or centrally (on the UDDI). Such federations may represent common interest groups of peers that band together to ensure that they provide added-value syndicated services to their customers. A *peer (service) syndication* seeks to promote in-demand services by offering sets of related services throughout the federation rather than on a single UDDI location, see Figure 1.

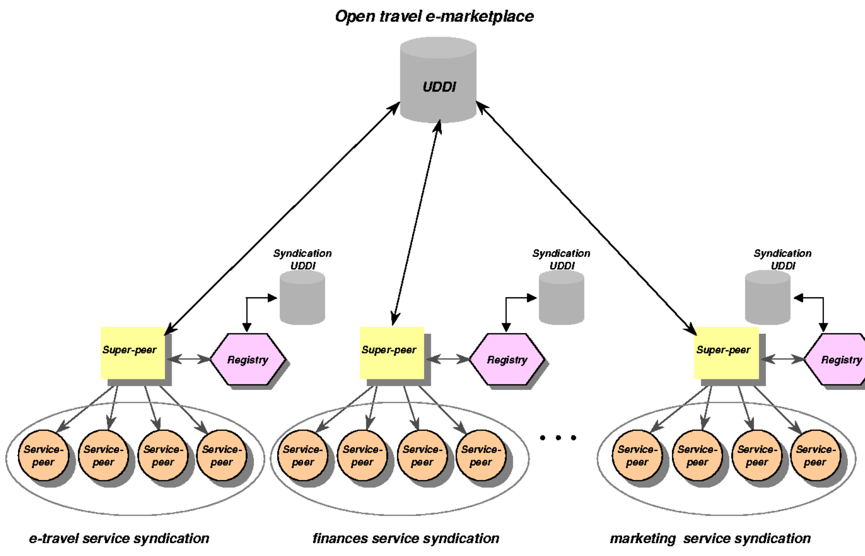


Fig. 1. Conceptual architecture of the P2P service network.

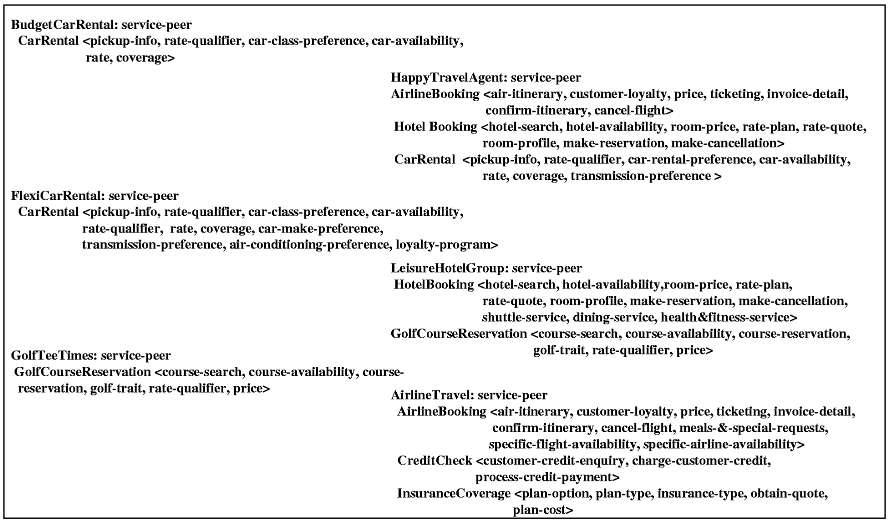
Two of the key concepts in a peer service syndication are the notions of *publication* and *subscription*. Publications are simple XML documents that name, describe and publish the existence of peers that act as service providers, while subscriptions also name, describe and publish the service requirements of peers that act as service requesters within a service syndication. Discovery within a service syndication becomes an issue of matching service subscriptions against service publications.

A peer syndication is formed for specific specialised areas of interest within an e-marketplace, e.g., e-travel, finances, marketing and so on. Service providers (peers) first publish their services on the e-marketplace UDDI and then they may join a service syndication. When joining the P2P web-service network, a peer first registers itself by publishing services it wishes to offer to other peers. Secondly, it may subscribe to services that it is interested in from other peers in the syndication. For each syndication a specific peer acts as *super-peer* by providing directory services to the peer syndication, see Figure 1. The registry of the super-peer contains among other things a syndication UDDI sub-directory. Whenever a peer joins a syndication the syndication UDDI receives a mirror copy its service publication from the e-marketplace UDDI.

The super-peer acts as an event-notification service that receives and stores the publications and subscriptions of the entire peer-syndication. To achieve this a super-peer manages a select set of meta-operations for peers, such as joining/leaving the network, publishing service publications, and service subscriptions.

Event-notification is a concept used for asynchronous coordination of distributed systems. The event notification service (super-peer) can carry out a *selection process* (on the basis of subscription/publication matching) to determine which of the published notifications are of interest to which of its peers, thus routing and delivering notification only to those peers that are interested.

To exemplify these points we introduce a service syndication scenario in the domain of e-travelling based on specifications of the open travel agency (OTA) [13]. OTA has specified a set of standard business processes, which use XML for structured data messages, for searching for availability and booking a reservation in the airline, hotel and car rental industry, as well as the purchase of travel insurance in conjunction with these services.



BudgetCarRental: service-peer
CarRental <pickup-info, rate-qualifier, car-class-preference, car-availability, rate, coverage>

HappyTravelAgent: service-peer
AirlineBooking <air-itinerary, customer-loyalty, price, ticketing, invoice-detail, confirm-itinerary, cancel-flight>
Hotel Booking <hotel-search, hotel-availability, room-price, rate-plan, rate-quote, room-profile, make-reservation, make-cancellation>
CarRental <pickup-info, rate-qualifier, car-rental-preference, car-availability, rate, coverage, transmission-preference >

FlexiCarRental: service-peer
CarRental <pickup-info, rate-qualifier, car-class-preference, car-availability, rate-qualifier, rate, coverage, car-make-preference, transmission-preference, air-conditioning-preference, loyalty-program>

LeisureHotelGroup: service-peer
HotelBooking <hotel-search, hotel-availability,room-price, rate-plan, rate-quote, room-profile, make-reservation, make-cancellation, shuttle-service, dining-service, health&fitness-service>
GolfCourseReservation <course-search, course-availability, course-reservation, golf-trait, rate-qualifier, price>

GolfTeeTimes: service-peer
GolfCourseReservation <course-search, course-availability, course-reservation, golf-trait, rate-qualifier, price>

AirlineTravel: service-peer
AirlineBooking <air-itinerary, customer-loyalty, price, ticketing, invoice-detail, confirm-itinerary, cancel-flight, meals-&-special-requests, specific-flight-availability, specific-airline-availability>
CreditCheck <customer-credit-enquiry, charge-customer-credit, process-credit-payment>
InsuranceCoverage <plan-option, plan-type, insurance-type, obtain-quote, plan-cost>

Fig. 2. Sample publications in the e-travel service syndication.

Figure 2 illustrates possible publications made by airline associations, car rental agencies, hotel corporations and leisure operators that have published their services as part of an e-travel service syndication. For reasons of brevity we show only six service peers in this figure. Each publication consists of three parts: the peer's (service provider's) name, e.g., `BudgetCarRental`, its WSDL `portType` name, e.g., CarRental, and the name of the operations contained in it. WSDL `portType`s contain the abstract definition of operations. All service providers in the marketplace (and syndication) use standard (unique) names for their port types and associated operations.

The goal of each super-peer is to send an update to its syndication to notify all potential subscribers about new publications (registrations) and deletions of registrations it receives. When a super-peer receives a new publish/subscribe event from a peer that wishes to join the network, e.g., `BudgetCarRental`, it performs two kinds of matching operations. The first matching operation matches the new peer's publication against all subscriptions that are relevant to it. If, for instance, `BudgetCarRental` has published information about CarRental rate qualifiers (rq), and car class preferences (ccp) then it could be matched with the subscriptions of the peers `HappyTravelAgent` and `FlexiCarRental` - assuming that these have subscribed to such information. The second matching operation matches all previous publications against the service subscriptions of this new peer. If, for instance, `BudgetCarRental` has subscribed to information relating to HotelBooking it can be matched with peers such as `LeisureHotelGroup` and `HappyTravelAgent` assuming that these two peers have published the information in which `BudgetCarRental` is interested. Whenever a match is detected, the super-peer forwards the peer publication information such as the services offered, their matching peers and their addresses to all the peers that have subscribed to their services. As a consequence, each peer contains high-level service descriptions as well as the addresses of the peers that have published information (services) that this peer has subscribed to. In this way the new peer is informed about existing peers whose publications match its subscription needs and can thus establish its own peer group within the syndication.

Each peer within a service syndication knows only the identity of the peers in its own syndication that match its own subscription needs. For instance, based on its previously stated subscriptions `BudgetCarRental` would form a syndication with peers such as `LeisureHotelGroup` and `HappyTravelAgent`. A peer can thus form its own *peer-acquaintance group* (PAG) within the syndication dynamically. Once peers have formed their own syndication they can work autonomously by exchanging services and information with each other without having to rely to their super-peer any more. Peers collaborate by propagating service requests to peers within their PAG to which they subscribe (henceforth named *acquainted peers*). Acquainted peers respond to a service request issued by a *local peer* in their syndication by returning a high-level UDDI description of their service content such as their `BusinessKey`, `Tmodel` structure and `bindindTemplate`. After receiving this information, the local peer invokes the `get_detail()` operations of the syndication UDDI API to retrieve more detailed information about the service port-types, elements and bindings of the services offered by its acquainted peers, see Figure 1.

The above P2P service network combines aspects of the directory services P2P model exemplified by Napster [12] and the "pure" P2P architecture exemplified by Gnutella and Freenet [9,6]. It follows a federated approach where a relatively small number of super-peers provide directory services to peer groups managed by a super-peer. The super-peer is used only for notification purposes when peers attempt to locate or communicate with each other. Each peer in the P2P service network builds up a (constantly changing) peer-group of other peers and stores locally some minimal information about them. Whenever a peer receives a request for service that it cannot fully satisfy it routes segments of it to appropriate peers within its peer-group for execution.

4 Publish/Subscribe in the P2P Web-Services Network

The publish/subscribe mechanism used in this paper is a *P2P communication protocol* that enables an exchange of asynchronous notifications between loosely coupled peers in a P2P network of web-services. This mechanism is partially based on an event notification scheme proposed for wide-area networks [2].

4.1 Service Publication and Subscription Matching

A peer publishes a set of WSDL port-types, each having a set of operations characterizing the service it offers. For a given service syndication we assume that the set of all possible port-type and operation names **PN** and **OP**, respectively, are well-defined. All publications are maintained by a super-peer known to all peers in the peer set **P**. The sets **PN**, **OP**, and **P** contain all permissible port type, operation, and peer names, respectively.

Organizing and manipulating publication contexts. A *port-type* is a pair (n, O) with $n \in \mathbf{PN}$ and $O \subseteq \mathbf{OP}$ and a *publication* can be expressed as a set of port-types. The information about publications and their publishing peers can be maintained in a matrix relating a set of peers with all their publications known in a specific service syndication at a particular time. We call this matrix a *publication context.* The peer names are represented by rows in the matrix, while the port-types are represented by its columns. A cross in row p and column (pt, O) indicates that peer p has published port-type (pt, O). Table 1 illustrates a publication context for the port-types and peers used in the example in Figure 2[1].

Publication contexts and the matching of a peer's subscription against a given publication context can be modelled mathematically in terms of formal concept analysis [8], which relies on the theory of ordered sets and complete lattices. Formally, a publication context $C := (P, Pt, I)$ consists of a set $P \subseteq \mathbf{P}$ of peer names, a publication, i.e., a set of port-types, $Pt \subseteq \{(n, O) \mid n \in \mathbf{PN} \wedge O \subseteq \mathbf{OP}\}$, and an incidence relation $I \subseteq P \times Pt$. The fact that a peer p has published a certain port-type (n, O) is expressed as $(p, (n, O)) \in I$.

For a set of peers in $Q \subseteq P$ the expression:

$$[Q] := \{pt \in Pt \mid (p, pt) \in I \text{ for all } p \in Q\} \tag{1}$$

computes the set of *port-types belonging to the peers in* Q. For instance, $[$`LeisureHotelGroup`$] = \{$`GolfCourseRes(cs,ca,cr,gt,rq,p)`, `HotelBk(hs,ha,rpr,rp,rq,rpf,mr,mc,ss,ds,hfs)`$\}$

A publication context can be extended or modified incrementally by means of only a few operations provided by each super-peer as follows:

[1] Due to space limitations, we henceforth represent in all tables and figures operations corresponding to the constructs in Figure 2 by their initial letters only.

Table 1. Example of a publication context for the travel syndication service depicted in Fig. 2

	AirlineBk(ai,cl,p,t,id,ci,cf)	AirlineBk(ai,cl,p,t,id,ci,cf,msr,sfa,saa)	CarRental(pi,rq,ccp,ca,r,c)	CarRental(pi,rq,crp,ca,r,c,tp)	CarRental(pi,rq,ccp,ca,r,c,tp,cmp,acp,lp)	CreditCheck(cce,ccc,pcp)	GolfCourseRes(cs,ca,cr,gt,rq,p)	HotelBk(hs,ha,rpr,rp,rq,rpf,mr,mc)	HotelBk(hs,ha,rpr,rp,rq,rpf,mr,mc,ss,ds,hfs)	InsuranceCov(po,pt,it,oq,pc)
AirlineTravel		×				×				×
BudgetCarRental			×							
FlexiCarRental					×					
GolfTeeTimes							×			
HappyTravelAgent	×			×				×		
LeisureHotelGroup							×		×	

`void addPeer(peer` p`):` The effect of this operation on a given publication context (P, Pt, I) is an extended context $(P \cup \{p\}, Pt, I)$. The corresponding matrix is extended by a new row labelled p, provided that a row labelled p does not exist already.

`void addPortType(peer` p`, portType` pt`):` The current publication context (P, Pt, I) is extended to yield $(P, Pt \cup \{pt\}, I \cup \{(p, pt)\})$ if $p \in P$, otherwise it remains unchanged. If a row labelled p exists in the matrix, the matrix will then be extended by a new column pt (if not already available) while a cross is introduced at the intersection of row p and column pt (provided that it does not exist).

`void addOperation(peer` p`, portTypeName` n`, opName` o`):` The effect of this operation on the current context (P, Pt, I) is a new context (P, Pt', I') with $Pt' := Pt \cup \{(n, O \cup \{o\}) \mid (n, O) \in Pt \wedge o \notin O \wedge (p, (n, O)) \in I\}$ and $I' := I \cup \{(p, (n, O \cup \{o\})) \mid (n, O) \in Pt \wedge o \notin O \wedge (p, (n, O)) \in I\}$. The matrix is extended by a new column labelled $n(o_1, \dots, o_n, o)$ for the column $n(o_1, \dots, o_n)$ related to row p, provided that o is different from $o_1, \dots, o_n$. In addition, a cross is introduced at the intersection of row p and the new column $n(o_1, \dots, o_n, o)$.

`void addPublication(peer` p`, portType` pt`):` This operation is defined by combining the operations `addPeer(peer` p`)` and `addPortType(peer` p`, portType` pt`)`.

`void deletePeer(peer p):` If $p \in P$ of the current context (P, Pt, I), the result is the context $(P - \{p\}, Pt', I')$ with $Pt - \{pt \mid (p, pt) \in I \wedge \not\exists (p', pt) \in I \text{ with } p \neq p'\}$ and $I' := I - \{(p, pt)\}$.

In the matrix we just delete row p and all columns pt that are related only to p.

`void deletePortType(peer p, portType pt):` The intended meaning of this operation is that peer p no longer publishes port-type pt. The modified contex is defined by (P, Pt', I') with

$$Pt' := \begin{cases} Pt - \{pt\} \text{ if } & \nexists (p', pt) \in I \text{with } p \neq p' \\ \text{otherwise } Pt \end{cases}$$

and $I' := I - (p, pt)$.

`void deleteOperation(peer p, portTypeName n, opName o):` The effect of this operation is a context in which operation o is removed from port-type pt published by peer p. More precisely, we obtain the new context (P, PT', I') defined as follows: $Pt' := Pt - \{(n, O) \mid (n, O) \in Pt \wedge o \in O \wedge \nexists (p', (n, O)) \in I \text{with } p \neq p'\} \cup \{(n, O - \{o\}) \mid (p, (n, O)) \in I \wedge o \in O\}$ and $I' := I - \{(p, (n, O)) \mid o \in O\} \cup \{(p, (n, O - \{o\})) \mid (p, (n, O)) \in I \wedge o \in O\}$ under the condition that p exists and p is related to any port-type $pt \in Pt$ that contains operation o. Otherwise the context remains unchanged.
This operation has the following effect on the matrix. The column that is labelled $n(op_1, \ldots, o, \ldots o_n)$ and has a cross at its intersection with row p is removed. A new column labelled $n(op_1, \ldots, o_n)$ is added and a new cross is added in the newly inserted column $n(op_1, \ldots, o_n)$.

Service searching and subscription. A search for services is typically triggered by a requester indicating the port-type and the associated set of operations that it is interested in. That is, a service request $r = (n, \{o_1, \ldots, o_n\})$ is a port-type name and a set of operation names. A peer p matches a service request r in a given publication context (P, Pt, I) if p publishes a port-type (n, O) that includes at least the operations contained in the request. Accordingly, we define a function $match(p, r) \equiv \exists (n, O) \in [\{p\}]$ such that $\{o_1, \ldots, o_n\} \subseteq O$ (see Definition 1 for the meaning of the symbol [ ]). All publishing peers that satisfy a request r form the *peer-acquaintences* $acq(r)$ group of the peer that issues a service request. Consequently, $acq(r) := \{p \in P \mid match(p, r)\}$.

In general, a peer seeks other peers that publish a coherent set of port-types and not just a single port-type. Consequently, we extend the above definitions to what we call a *subscription.* A subscription S is defined, just like a publication, as a set of pairs $(s, O))$ with $s \in \mathbf{PN}$ and $O \subseteq \mathbf{OP}$. We say that a peer p *matches a subscription* S if it matches all requests in S, i.e., $match(p, S) \equiv \forall r \in S : match(p, r)$. The *peer-acquaintances* $acq(S)$ of a subscription S with respect to a publication context (P, Pt, I) and a subscription request r are defined as:

$$acq(S) := \bigcap_{r \in S} acq(r). \tag{2}$$

Table 2 shows simple examples of peer-subscriptions and their resulting peer-acquaintances. This table shows that subscription 2 extends Subscription 1 by

Table 2. Examples of peer-subscriptions and their acquaintances.

Example	*Subscription*	*Acquaintances*
Subscription1	{`CarRental(rq,ccp,r,c`}	{`BudgetCarRental`, `FlexiCarRental` }
Subscription2	{`CarRental(rq,ccp,r,c,tp,cmp,acp`}	{`FlexiCarRental`}
Subscription3	{`HotelBk(hs,ha,rpr,rp,rq,mr)`, `GolfCourseRes(cs,ca,rq,p)`}	{`LeisureHotelGroup`}
Subscription4	$\emptyset$	P
Subscription5	{`CarRental(pi,ccp)`, `HotelBk(hs,ha)`}	$\emptyset$
Subscription6	{`AirlineBk()` }	{`AirlineTravel`, `HappyTravelAgent`}

requesting additional operations to be matched. These additional operations are found only in the `CarRental` port-type published by peer `FlexiCarRental`, i.e., a subset of the acquaintances of subscription 1. We also observe that the port-type of Subscription 2 is a subtype of the port-type of Subscription 1. Therefore, we define a *subtype relationship* "$\leq$" between port-types $pt = (n, O)$ and $pt' = (n', O')$ with $n, n' \in \mathbf{PN}$ and $O, O' \in \mathbf{OP}$ that is induced by the subset relationship among the operations as follows:

$$pt' \leq pt \equiv (O \subseteq O' \wedge n = n')$$

We also notice that an empty subscription like Subscription 4 is matched by all peers, and Subscription 5 yields an empty set of acquaintances because the requested combination of port-types is not supported by any peer in the syndication. Subscription 6 is matched by all peers advertising the port-type named `AirlineBk` as it contains no operations. This signifies that the subscriber is obviously satisfied with any possible operation included in the requested port-type.

The above considerations imply that subtyping relationships between logically associated port-types should be first introduced at the publication context and then be used to match potential subscriptions. We can extend a given publication context (P, Pt, I) to the context $(P, \overline{Pt}, \overline{I})$, by taking all possible subtype relationship between port-types carrying the same port-type name into account. This is defined as:

$$\overline{Pt} = Pt \cup \{(n, Q) \mid (n, O) \in Pt \wedge Q \subseteq O\} \tag{3}$$

and

$$\overline{I} = I \cup \{(p, (n, Q)) \mid (p, (n, O)) \in I \wedge Q \subseteq O\} \tag{4}$$

In the matrix representation of a publication contexts this means to add: (1) all necessary columns (n, Q) for each published port-type (n, O) such that $(n, O) \leq (n, Q)$ and (2) a cross at the intersection of row p with each new column (n, Q) provided that a cross exists at the intersection of row p with column (n, O), for each $(n, O) \leq (n, Q)$. Table 3 is the extended table constructed from Table 1 by adding all appropriate port-type subtype relationships. Additional columns (not all of which are shown) are marked in a light grey colour.

Table 3. Example of a supplemented publication context for the travel domain depicted in Fig. 2.

	AirlineBk()	AirlineBk(ai)	...	AirlineBk(ai,cl,p,t,id,ci,cf)	AirlineBk(ai,cl,p,t,id,ci,cf,msr)	...	AirlineBk(ai,cl,p,t,id,ci,cf,msr,sfa,saa)	CarRental()	...	CarRental(pi,rq,ca,r,c)	CarRental(pi,rq,ccp,ca,r,c)	CarRental(pi,rq,crp,ca,r,c)	CarRental(pi,rq,ca,r,c,tp)	CarRental(pi,rq,crp,ca,r,c,tp)	CarRental(pi,rq,ccp,ca,r,c,tp)	...	CarRental(pi,rq,ccp,ca,r,c,tp,cmp,acp,lp)	...
AirlineTravel	×	×	×	×	×	×	×											
BudgetCarRental								×	×	×	×							
FlexiCarRental								×	×	×	×		×		×	×	×	
GolfTeeTimes																		
HappyTravelAgent	×	×	×	×				×	×	×		×	×	×				
LeisureHotelGroup																		

We can simplify our definition of the acquaintances of a subscription $S \subseteq \overline{Pt}$ by using the extended publication context and matrix $(P, \overline{Pt}, \overline{I})$ as follows:

$$acq(S) = [S] := \{p \in P \mid (p, pt) \in \overline{I} \text{ for all } pt \in S\}. \tag{5}$$

where $[S]$ determines the set of all peers that publish all port-types in S. We shall henceforth always refer to the extended form of a publication context.

Subscriptions are maintained the same way as publications are maintained. In addition, any change to a peer's publication context must be automatically mirrored against the set of known subscriptions which are maintained in a subscriber's table. Subscribers provide a call-back method that is invoked from the super-peer to inform them when they are affected from any changes that invalidate or extend a subscription that they made. A subscription is invalidated, for instance, if:

– an operation that a subscriber requested from a port-type is deleted from that port-type,
– the peer to which the subscription was bound is deleted, or
– a port-type that is a subtype of a requested port-type is deleted.

A subscription is extended if a new subtype of a subscribed port-type is added or a new operation is added to a super-type of requested port-type such that the extended port-type now matches the requested port-type.

Structural organisation of publications and subscriptions. The acquaintances of a subscription can be computed by traversing the extended matrix and selecting all the peers that satisfy a subscription. A more elegant way to organize publication contexts and subscriptions and determine matching acquaintances is by means of a *concept lattice* $\mathbf{C}(P, Pt, I)$. This lattice is derived automatically from a given publication context (P, Pt, I) using an efficient algorithm described in [7]. This algorithm relies on the notion of a *formal concept* of a context (P, Pt, I), which is defined as a pair (Q, T) with $Q \subseteq P, T \subseteq Pt$, $[Q] = T$ and $[T] = Q$ (see definition 1 and peer-publish-all-relations, respectively.). Q is called the *extent* and T is the *intent* of the concept (Q, T). If two concepts such as (Q_1, T_1) and (Q_2, T_2) are two concepts of a given context, then (Q_1, T_1) is called a *subconcept* of (Q_2, T_2) if $Q_1 \subseteq Q_2$ or, equivalently, if $T_2 \subseteq T_1$.

Figure 3 illustrates the concept lattice of the complete sample publication in Table 1. This figure includes all missing port-types and operations found in Figure 2. The super-peer organises the publication space around this type of concept lattice and stores it in its own local registry.

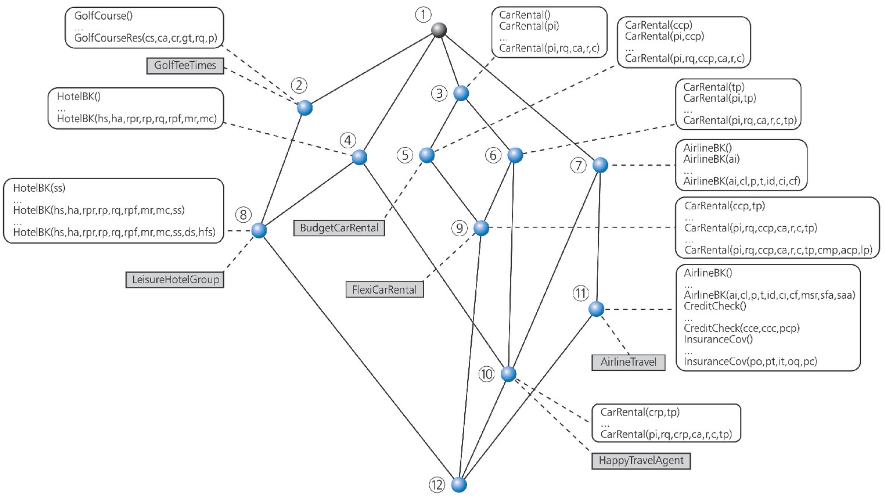


Fig. 3. Concept lattice including subtype relationships among published services.

There are two types of labels attached to the nodes in this publication lattice: port-types and peer names (shaded rectangles). Each node represents a formal concept relating its extent, i.e., a set of peer names, with its intent, i.e., a set of port-types published by all peers in its extent. All port-types reachable via upwards paths from a node in the lattice determine the node's intent. For instance, the intent of node 10 is the sum of the port-types listed in nodes 1, 3, 4, 6, and 7. Conversely, all peers reachable downwards from a node in the lattice form the extent of that node. The peers in the extent of a given node are those peers that have published the port-types associated with its intent. Consider, for instance, node 6: the port-types `CarRental()`,`CarRental(pi)`, ..., `CarRental(pi,rq,cq,r,c,tp)`, which form the intent of node 6, are published by the peers associated with node 9 and 10, i.e, `FlexiCarRental` and `HappyTravelAgent`, which form its intent. The top node of the lattice represents the concept relates all the peers in a given publication context with the port-types they all share in common (in our example there are none therefore the concept is $(P, \emptyset)$). The bottom node of the lattice relates the sum of all port-types in a publication context with those peers that publish this sum (in our example there are none, therefore the concept associated with the bottom node is $(\emptyset, Pt)$).

The collection of subscriptions in a syndication are organised in a similar fashion in a subscription lattice. In this way we can use a uniform mechanism to determine subscription/publication matches and which subscribers are affected by a change to the publication lattice.

4.2 Peer Group Formation

A peer group is formed by applying the *match* function on a peer's subscription against a publication context C. Suppose that the very first subscription of the peer `BudgetCarRental` is: $S := \{\texttt{CarRental(tp)}, \texttt{HotelBooking(hs,..,mc))}\}$. A *matched-subscription lattice* for `BudgetCarRental`'s subscription, shown in Figure 4, is generated by matching subscription S against the complete publication lattice of the super-peer, shown in Figure 3. The matched-subscription lattice is a true subset of the extended publication lattice stored at the super-peer and indicates the peers and matching concepts that conform to `BudgetCarRental`'s subscription. This figure shows that `BudgetCarRental`'s subscription has resulted in forming a lattice with the peeers: `HappyTravelAgent`, `LeisureHotelGroup` and `FlexiCarRental`. In this matched subscription lattice `LeisureHotelGroup` provides the additional hotel functionality requested but not the car rental functionality, `FlexiCarRental` provides the additional car rental functionality requested but not the hotel functionality, while `HappyTravelAgent` provides both requested functionalities. In the first instance when a peer has made its initial subscription, the matched-subscription lattice is stored locally at this peer's site and its forms its *local subscription lattice.* This local subscription lattice can be used in the future to locate relevant peer acquaintances when attempting to process a service request without having to resort to the super-peer any more. A local subscription lattice grows as a peer gets notified of new subscriptions.

A peer's publication lattice is thus fluctuating with peer relationships breaking and reforming dynamically based on the peer's interests and the number of peers entering or leaving its group as a result of this.

Whenever a *service request* is posed at a local peer, its set of subscriptions needs to be evaluated to determine whether it can support this new request. In case that execution of the new request is not fully supported locally, then a new subscription (for this peer) reflecting the missing information will be generated from the service request and will be sent to the super-peer.

When a peer decides to leave the P2P network, a notification will be sent from the super-peer to all relevant peers, which in turn will update their lo-

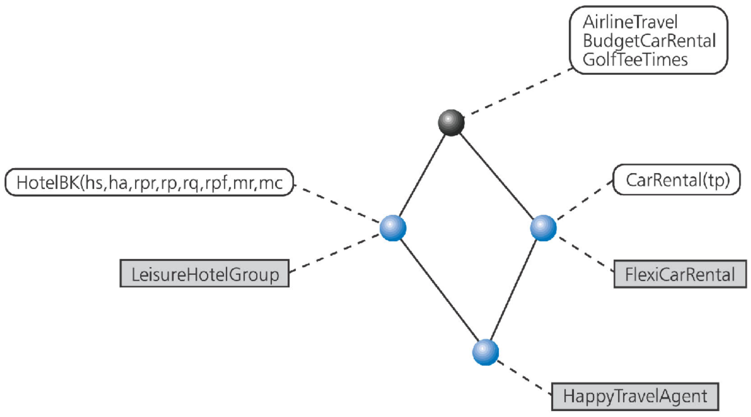


Fig. 4. A matched-subscription lattice for `BudgetCarRental`

cal subscription lattices. This happens in accordance with the lattice updating operations defined in the previous subsection.

5 Service Request Processing

When a service request arrives, the local peer will use its local subscription lattice as a point of reference to resolve a set of acquainted peers in its PAG to which the request should be routed. The response to a request includes descriptions of the service content that the acquainted peers provide in connection to the sub-request they receive.

To understand how service requests are processed assume that `BudgetCarRental` that has formed a PAG with peers `HappyTravelAgent`, `LeisureHotelGroup` and `FlexiCarRental`. Further assume that this peer needs to handle a sub-request that requires hotel booking functionality, e.g., search for a particular type of hotel that is not supported locally. After receiving this request `BudgetCarRental` determines (on the basis of its local subscription lattice, shown in Figure 4) that its acquainted peers `HappyTravelAgent` and `LeisureHotelGroup` can handle this type of request. Subsequently, the peer `BudgetCarRental` requests technical information about its two acquainted peers from the UDDI using the `get_Detail` operations of the UDDI enquiry API. The `get_Detail` operations are used to retrieve technical details such as access information required to invoke a service. This technical information describes the format input messages should be sent in, what protocols are appropriate, what security is required, and what form of a response will result after sending input messages. More specifically, the `get_ServiceDetail`, `get_bindingDetail`, and `get_tModelDetail` functions of the UDDI API are invoked to retrieve the above technical details that are required to interact with a service endpoint.

Now assume that after receiving this technical information the local peer `BudgetCarRental` has decided to invoke one (or both) of its two acquainted peers HappyTravelAgent and LeisureHotelGroup. Also assume that the service requester is interested in hotels within 5 kms North West of Schiphol airport. This request can be expressed as shown in Figure 5 on the basis of the standard Hotel Search Request message provided by the OTA specifications for hotel industry messages (see section-5 of OTA 2001 Message specifications). This message provides the ability to search for a list of hotel properties that meet certain criteria, and supply information in return that is related to the specific request. Geographic data, such as proximity to a specific location, landmark, attraction or destination point, could also be used to constrain the summary response to a limited number of hotels.

```
<Request HotelSearchRQ xmlns="http://www.opentravel.org/OTA"
       xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
       xsi:noNamespaceSchemaLocation="OTA_HotelSearchRQ.xsd">

       <HotelSearchCriterion Type="Area" MatchType="Exact"
                                      ImportanceType=="Mandatory" >
         <HotelSearchValue>
             <RefPoint Distance="5" DistanceMeasure="km"
              Direction="NW">Schiphol Airport </RefPoint>
         </HotelSearchValue>
      </HotelSearchCriterion>
</Request>
```

Fig. 5. Sample service request.

In Figure 5 the statement < HotelSearchRQ > identifies the request as targeting hotel property data. This statement may have one to many < HotelSearchCriterion > child elements that identify a single search criterion by means of criteria types. A < MatchType > attribute indicates whether the match to a string value must be exact or only partial. The < ImportanceType > attribute is used to allow the responding web-service implementation to search for appropriate hotels and respond to preference criteria in the order of importance to the service client. This construct indicates whether the input criterion is mandatory, of high, medium or low priority. The < HotelSearchValue > element is a required child element of < HotelSearchCriterion > that contains the values expected by the Type attribute.

The response to the request in Figure 5 returns a list of hotel properties that meet the criteria of the request. A sample response to the request of Figure 5 is found in Figure 6.

6 Related Work

Recent work in content-based search include *content-addressable networks* – where the content of queries is used to efficiently route messages to the most relevant peers – such as CAN [14], Chord [4], and Pastry [5] as well as some variations of publish/subscribe networks [11]. These content-based P2P networks place emphasis on discovery of content rather than on a logical organization of the information space and on establishing relationships between constructs in

```
<?xml version="1.0" encoding="UTF-8"?>
<OTA_HotelSearchRS xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="OTA_HotelSearchRS.xsd" xmlns="http://www.opentravel.org/OTA">
<Success/>

<HotelSearchRecord HotelName="Hilton Hotel" Relevance="100">
     <HotelReference ChainCode="HH" BrandCode=".." HotelCode=".."/>
      <LocationDescription> 3.5 kms NW of Schiphol Airport</LocationDescription>
      <SearchValueMatch Match="true">Deluxe</SearchValueMatch>
      <MarketingText>Pool, Spa, and Health Club on premises</MarketingText>
</HotelSearchRecord>
    ... ... ...
<TotalReturns> 5 </TotalReturns>
</OTA_HotelSearchRS>
```

Fig. 6. Sample service response.

the publication space. A major difference between our approach and these activities is that our approach structures the publication and subscription space in concepts lattices that logically organize the publication/publication content by establishing semantic links and relationships between content concepts such as port-type and operation names.

On the industrial research side, work that comes closer to the research reported herein is Sun Microsystem's Project Juxtapose (http://www.jxta.org) – usually referred to as JXTA [15]. JXTA is not based on a publish/subscribe and event notification scheme. Its search peers act as hubs and can register with other hubs as information providers to field queries from other peers based on arbitrary content description registrations. JXTA tends to centralize registration/subscription information and control in hubs, whereas in our approach peers register only high-level information about themselves, such as their name, address and names of the service elements they are willing to share with other peers, with a super-peer. Moreover, unlike JXTA peers do not use the super-peer to locate each other or communicate with one another. Instead, each peer builds up a (constantly changing) peer-group of other peers and stores some minimal information about them.

7 Summary

In this paper we highlighted key intersect points that enable using service-oriented and peer-to-peer-computing technologies together and presented an architectural approach and formal framework towards unifying them.

We introduced a federation of UDDI-enabled peer registries, which operate in a decentralized fashion, rather than requiring each peer to publish their own service descriptors locally or centrally (on the UDDI). Federations are collections of behaviourally similar cooperating peers that provide a common set of circumstances and that band together to ensure that they provide added-value syndicated-services to their customers. Peer federations use a publish/subscribe model that enables the loosely coupled exchange of asynchronous notifications and an efficient mechanism based on concept-lattices to match notifications to subscribers and route notifications from publishers to interested subscribers.

There are several advantages that the publish/subscribe model for P2P networks of web-services offers. These include simplicity and ease of use, openness, extensibility, scalability, and semantic-based request routing.

References

1. S. Botros and S. Waterhouse, "Search in JXTA and other Distributed Networks," Proc. 2001 Int'l Conf. Peer-to-Peer Computing, 2001;
available online at http://people.jxta.org/stevew/BotrosWaterhouse2001.pdf.
2. A. Carzaniga, D. Rosenblum, and A. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service", in ACM Trans on Computer Systems, Vol. 19, No. 3, pp. 332–383, 2001.
3. Curbera, F., Goland, Y., Klein, J., Leyman, F., Roller, D., Thatte, S., and Weerawarana, S., "Business Process Execution Language for Web Services(BPEL4WS) 1.0," August 2002, available at http://www.ibm.com/developerworks/library/ws-bpel.
4. F. Dabek, et. al. "Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service", http://pdos.lcs.mit.edu/chord2001.
5. P. Druschel and A. Rowstron "Pastry: Scalable Distributed Object Location and Routing for Large Scale Peer-to-Peer Systems", ACM SIGCOMM, 2001.
6. Freenet Home Page. http://freenet.sourceforge.com
7. Algorithmen zur formalen Begriffsanalyse. In: B. Ganter, R. Wille, and K.E. Wolff (eds), *Beiträge zur Begriffsanalyse.* B.I.-Wissenschaftsverlag, Mannheim, 1987, pp. 241–254
8. B. Ganter, R. Wille. *Formal Concept Analysis.* Springer 1999
9. Gnutella Development Home Page. http://gnutella.wego.com
10. L. Gong. "Project JXTA: A Technology Overview", available at
http://www.jxta.org/project/www/white_papers.html.
11. D. Heimbigner "Adapting Publish/Subscribe Middleware to Achieve Gnutella-like Functionality", ACM Symposium on Advanced Computing (SAC), 2001.
12. www.napster.com, 2001.
13. Open Travel Alliance (OTA), "Document 2001C Specifications", available at http://www.opentravel.org, 2001.
14. S. Ratnasamy, et. al. "A Scalable Content Addressable Network", ACM SIGCOMM, 2001.
15. S. R. Waterhouse, D. M. Doolin, G. Kan and Y. Faybishenko, "JXTA Search: a Distributed Search Framework for Peer-to-Peer Networks" in IEEE Internet Computing, vol. 6, pp. 68–73, 2002.