Service-Komposition von Reiseprozessen mittels Graphtransformation

Jörg Daubert[1], Erwin Aitenbichler[2], Stephan Borgert[2]

[1]Fachbereich Informatik, Technische Universität Darmstadt
[2]Telecooperation Group, Fachbereich Informatik, Technische Universität Darmstadt
{daubert|erwin}@informatik.tu-darmstadt.de,
borgert@tk.informatik.tu-darmstadt.de

Zusammenfassung In dieser Arbeit wird ein dezentrales Verfahren zur Planung von Reiseprozessen vorgestellt. Transportdienstleister bieten ihre Dienste über einen Service-Marktplatz an und können mit Hilfe der Unified Service Description Language (USDL) effektiv vorselektiert werden. Der Reiseprozess wird durch schrittweise Verfeinerung und Graphtransformation erstellt. Auf diese Transformationen können Dienste direkt Einfluss nehmen. Das macht unser Verfahren im Gegensatz zu zentralen Planungsansätzen flexibel, offen und erweiterbar.

1 Einleitung

In dieser Arbeit wird ein neues, dezentrales Verfahren zur intermodalen Reiseplanung vorgestellt, das auf aktuellen Internet-of-Services (IoS) Technologien [7] basiert. Transportdienstleister können beliebige Modalitäten (Flug, Bahn, Bus, Taxi, ...) anbieten und stellen diese über Softwaredienste (Services) bereit, welche für Kunden über einen Marktplatz ansprechbar sind.

Dabei müssen prinzipiell die folgenden Probleme gelöst werden: *Finden von Diensten*, *Routing*, und *Scheduling*. Zunächst müssen die Modalitäten und Dienstanbieter ausgewählt werden, die in Frage kommen. Danach befasst sich *Routing* mit dem Finden der optimalen Route zwischen zwei Stopps. Anbieter, die auf Basis eines Fahrplanes operieren, schränken die verfügbaren Abfahrts- und Ankunftszeiten ein. Unter Berücksichtigung dieser Constraints befasst sich *Scheduling* mit der Erstellung eines optimalen Zeitplanes.

Existierende Ansätze lassen sich im Wesentlichen in zwei Kategorien einteilen: Einerseits existieren Systeme mit guten Lösungen für das Routing- und Scheduling-Problem, die allerdings auf zentral gespeicherten Modellen basieren. Das stellt aber eine in der Praxis kaum realisierbare Idealvorstellung dar, da Transportdienstleister die Hoheit über ihre Daten (u.a. gerichtlich) verteidigen. Eine aktive Einbeziehung in die Planung ermöglicht außerdem die bessere Nutzung von domänenspezifischem Wissen. Andererseits existieren SOA-basierte Systeme, die meist Dienste nur auf Grund ihrer technischen Schnittstellen auswählen. Dies ist ineffizient, da im Service Discovery eine wesentlich bessere Vorselektion von Diensten erreicht werden kann.

Im Folgenden stellen wir unseren Ansatz zur intermodalen Reiseplanung vor, der es erlaubt, das Navigations- und Scheduling-Problem in offenen Service-Märkten zu lösen.

Der Rest des Papers ist wie folgt aufgebaut. In Abschnitt 2 werden zunächst verwandte Arbeiten diskutiert. In Abschnitt 3 wird die Systemarchitektur vorgestellt, danach die Planungsmethode in Abschnitt 4. Abschnitt 5 beschreibt die Implementierung. Der Artikel schließt mit der Zusammenfassung in Abschnitt 6.

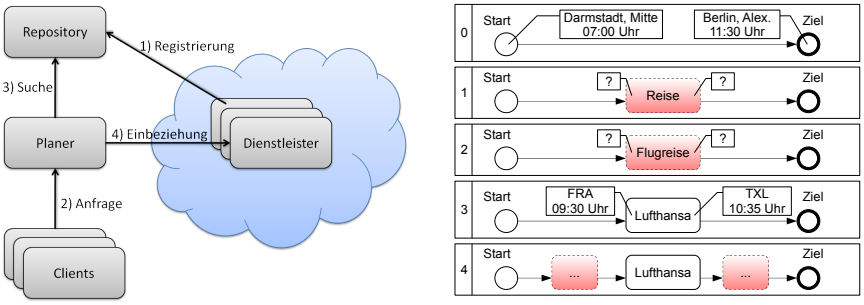
2 Verwandte Arbeiten

Graphenbasierte Modellierung mit mehreren Modalitäten wird in [8] vorgeschlagen. Hierbei werden jeweils eigene Kanten für jede Modalität verwendet. Zum Bestimmen von Routen wird eine an SQL angelehnte Abfragesprache vorgeschlagen. Der Ansatz adressiert primär das Routingproblem, berücksichtigt aber Abfahrtszeiten nur eingeschränkt. Im Rahmen des *iTransIT*-Frameworks [13] wird ein gemeinsames Datenformat für Modalitäten beschrieben, das *Common Data Model*. Es dient als Abstraktionsschicht, die über Geo-Datenbanken gelegt wird. Ein Reiseplaner ist durch den *Smart Traveler Information Service (STIS)* [9] realisiert. Für die Routenberechnung werden einzelne, logische Subgraphen für jede Transportmodalität verwendet. Jedoch wählt der Benutzer zuerst eine Modalität aus, danach wird eine Route auf dem entsprechenden Graphen mittels eines Kürzeste-Wege-Algorithmus gesucht, verbleibende "Lücken" werden dann mit weiteren Modalitäten geschlossen. STIS adressiert ebenfalls primär das Routing- und nicht das Schedulingproblem.

Ontologiebasierte Modellierung wird in [12] und [16] beschrieben. In [12] wird eine Reise als eine Reihe von geordneten *stop points* zwischen Start und Ziel modelliert. Lösungen werden mittels einer inferenzbasierten Ontology-Engine ermittelt, die zusätzlich auf Geo-Datenbanken zugreift. Unterschiedliche *journey patterns* können verwendet werden, um z.B. Routen mit "wenig Fußweg" oder mit "überdachten stop points" zu finden. In [16] wird eine Datenmodellierung mit dem Protégé-Werkzeug und eine Auswertung mit Hilfe des Jena-Frameworks vorgenommen. Die vorgestellte Evaluation ist mit nur 25 Elementen sehr klein.

Andere Ansätze wie [4] setzen auf **Constraint Programming**. Eine Reise besteht aus *tasks*, welche zu *templates* (etwa "tip", "fly") zusammengefasst werden können (Abstraktion). Je ein Teil der Reise wird herausgegriffen, Alternativen verglichen und dem Benutzer zur Auswahl gegeben. Das Constraint-Netzwerk umfasst alle Nebenbedingungen und Berechnungen, durch die Templates wird die Komplexität übersichtlich gehalten. Die Daten stammen von einer Reihe Agenten, etwa Wrapper und Screenscraper für Fahrplanauskünfte. Einerseits muss der Nutzer hier bei jedem Schritt aktiv werden und eine Wahl treffen, andererseits sind dem Aufbau einer Reise durch statische Templates enge Grenzen in der Flexibilität gesetzt [5].

SOA-basierte Systeme werden in [14] und [6] vorgestellt. Der in [14] vorgestellte Dienst ermittelt die günstigste Reise zwischen zwei Städten mittels eines Service-Mashups. Aufgrund beschränkter Granularität und kaum berück-



(a) Systemarchitektur (b) Schrittweise Prozessverfeinerung

sichtigter zeitlicher Nebenbedingungen eignet sich dieses Vorgehen kaum für ein allgemeineres Reiseproblem. Ähnlich ist *Self-Serv* [6] mit dem *Complete Travel Planning Service*, einer P2P-basierten Methode zur Web-Service Orchestrierung. Anhand eines State-Charts wird ein Reiseprozess erzeugt und nur auf Basis der Schnittstellen werden passende Services (etwa Flugbuchung) gewählt.

3 Architektur und Dienstbeschreibung

Der hier vorgestellte Ansatz zur automatischen Reiseplanung basiert auf IoS-Technologien und der Service-Beschreibungssprache USDL, die im Rahmen des Theseus/TEXO-Projekts [7] entwickelt wurden. Ziel von USDL (Unified Service Description Language) [15] ist es, eine umfassende Beschreibung zu schaffen, mit welcher zukünftig Dienstleistungen auf IoS-Marktplätzen angeboten und gefunden werden können. Eine wesentliche Neuerung von USDL ist der Einbezug nicht-technischer Eigenschaften von Diensten (“business”, “operational”). Somit können Ort und Zeit der Diensterbringung, sowie weitere nicht-funktionale Eigenschaften beschrieben werden.

Die Architektur ist in Abbildung 1a dargestellt und unterscheidet vier Arten von Teilnehmern: *Service Repository*, *Planer*, *Dienstleister* und *Clients*. Der Ablauf gestaltet sich wie folgt: Reisedienstleister beschreiben ihre Dienste mit USDL, also an welchen Orten diese in Anspruch genommen werden können, sowie Webservices zur Planung, und hinterlegen diese im Repository (1). Schickt ein Client eine Reiseplanungsanfrage an den Planer (2), erzeugt dieser mittels Graphtransformation einen Reiseprozess und fragt dabei die zu verwendenden Dienste am Repository ab (3). Dienste können dann vom Planer aktiv mit einbezogen werden (4).

4 Planung von Reiseprozessen

Das Ergebnis der Planung ist ein Reiseprozess, der detailliert beschreibt, wie der Nutzer vom Startort zum Zielort reisen kann. Dieser Prozess könnte später von einer Assistenzanwendung auf einem mobilen Gerät ausgeführt werden und dem Benutzer Navigationsanweisungen geben. Zur Modellierung, Darstellung und

Ausführung von Reiseprozessen verwenden wir Methoden aus dem Geschäftsprozessmanagement (BPM).

Das Prozessmodell basiert auf der Sprache PASS (Parallel Activities Specification Scheme) [11]. Unser Verfahren könnte ebenfalls zusammen mit anderen Sprachen, wie z.B. BPEL, angewendet werden. PASS erfüllt allerdings alle unsere Anforderungen und es kann viel an unnötiger Komplexität vermieden werden. Im Weiteren wurde in einer früheren Arbeit eine Engine entwickelt, die PASS-Prozesse auf mobilen Geräten ausführen kann [2]. Damit können Anwendungen zur mobilen Navigationsunterstützung des Benutzers erstellt werden.

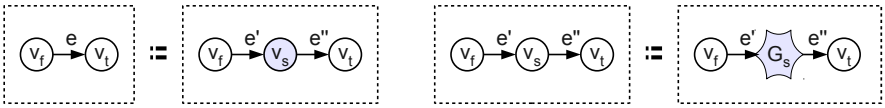
Aus der Sicht des Planers betrachten wir den Prozess zunächst abstrakt als Graphen $G = (V, E)$. Die Knoten V in diesem Graphen sind Aktivitäten, die durch unterschiedliche Dienstleistungen erbracht werden, oder Pseudoknoten, wie Start, Ziel, Split, Join, etc. Insbesondere entsprechen die Knoten also nicht räumlichen Orten, wie oftmals in Wegfindungsproblemen verwendet, sondern vielmehr Diensten in einem Prozess. Die Kanten E beschreiben mögliche Übergänge, also die zeitliche Abfolge von Aktivitäten.

Knoten sind mit Kontextinformationen attributiert, insbesondere sind dies Ort und Zeit. Diese Attribute existieren zweimal pro Knoten, nämlich für den geplanten Beginn der Aktion, sowie dem Ende. Weiterhin können alle Nicht-Pseudoknoten mit einer in USDL beschriebenen Dienstleistung versehen werden. Verwendet man einen Graphen mit ausgezeichnetem Start- und Zielknoten als Reiseprozess, dann ergibt sich mit jedem linearisierten Pfad zwischen Start und Ziel ein Reiseplan, der angibt, wann und wo welche Dienstleistung genutzt werden soll. Durch parallele Teilpfade lassen sich mehrere Alternativen ausdrücken, etwa dass ein Bus oder alternativ wenige Minuten später eine Straßenbahn verwendet werden kann. Durch einen derart gestalteten Graphen können auch komplexe Anfragen ausgedrückt werden, indem weitere Knoten für einen Hotelaufenthalt oder gewünschte Zwischenaufenthalte in den Ausgangsgraphen eingefügt werden.

Die Durchführung einer Reiseplanung erfolgt durch Anwendung einer Reihe von Regeln zur Transformation des Prozessgraphen. Die Kernidee dabei ist, mit einem sehr einfachen Graphen zu beginnen und diesen schrittweise zu verfeinern, also die Reise auszugestalten (Abbildung 1b). Der Graph wird mit dem Java-Framework *Graph Rewrite Library (GRL)* [1] bearbeitet. Graphsuchen und -ersetzungen werden dabei in der Sprache RDL (Rule Description Language) formuliert. Eine einfache Produktionsregel lautet z.B. wie folgt:

```
P() :- |F:Node,e:Edge,T:Node| :- F-e->T & T.startTime!=null
    := |S:Node,f:Edge| S=new Node(), f=new Edge(), F-e->S-f->T;
```

Die linke Seite der Regel (LHS) beschreibt das Muster, das im Graphen gefunden werden soll. In diesem Beispiel wird nach Belegungen der Variablen `F`, `e` und `T` gesucht, die zwei Bedingungen erfüllen: Der Pfadausdruck verlangt, dass `F` und `T` direkt durch die Kante `e` verbunden sind. Die folgende Bedingung überprüft, ob das `startTime`-Attribut von `T` gesetzt ist. Die rechte Seite der Regel (RHS) beschreibt die Transformation. Hier wird ein neuer Knoten `S` und eine neue Kante `f` eingefügt. Da GRL auch den Aufruf von Java-Methoden unterstützt,



(a) Einfügen (b) Substitution

Abbildung 1: Graphtransformationen

können Transformationen auch alternativ in Java implementiert werden. Das ist für komplexe Ersetzungen oft hilfreich.

Zu Beginn besteht der Graph nur aus dem Start- und Zielknoten, sowie einer Kante dazwischen. Zur schrittweisen Verfeinerung des Reiseprozesses dienen nun die folgenden drei Grundkonzepte: *Einfügen*, *Substitution* und *Adaption*.

Einfügen: Beim Einfügen wird im Repository ein Dienst gesucht, der die Transportlücke zwischen v_f (from) und v_t (to) möglichst gut schließt. Hierbei werden Ortsinformationen aus USDL-Beschreibungen ausgewertet. Weitere Kriterien, wie die aktuelle Komplexität des Graphen, sind möglich. Auch Vorlieben des Nutzers sind denkbar. Der neue Knoten v_s repräsentiert dann diesen Dienst. Kante e wird durch zwei neue Kanten e' und e'' ersetzt, deren summierte verbleibende Transportlücke kürzer als die von e sein muss. Mehrere Alternativen (parallele Pfade) sind ebenfalls möglich. Abbildung 1a zeigt diese Transformation.

Substitution: Im Falle der Substitution wird ein Knoten v_s, dessen USDL-Beschreibung einen Webservice zur Substitution umfasst, durch einen Subgraphen G_s ersetzt. Damit kann v_s an einen anderen Dienst delegieren, z.B. kann der abstrakte Knoten *Flugreise* durch den konkreten Anbieter *Lufthansa* ersetzt werden, oder G_s kann als Template für einen komplexen Subgraphen dienen. Der Anbieter *Lufthansa* kann etwa beschreiben, dass dieser Teil der Reise aus Check-in, Gepäckaufgabe, Boarding, Flug, ... besteht. Ein Dienstleister kann somit selbst die Transformation bestimmen, und damit den Reiseprozess entscheidend beeinflussen. Der Ansatz wird damit auch offen hinsichtlich beliebiger, neuer Transportmodalitäten. Dieses Vorgehen wird in Abbildung 1b illustriert.

Adaption: Die Adaption ist die schwächste Form der Transformation und wirkt sich nur auf die Attributierung eines Knotens aus. Auch hier wird ein durch die USDL-Beschreibung gegebener Webservice abgefragt. Sinnbildlich kann man diesen als Fahrplanauskunft betrachten. Dieses Konzept erlaubt die Handhabung von fahrplanbasierten und nicht-fahrplanbasierten Transportdiensten. Bei fahrplanbasierten Diensten erfolgt eine Anpassung an die Zeiten. Vor der Adaption könnte man etwa von einer unbestimmten Busfahrt sprechen, danach von einer festen Verbindung mit Haltestellen und Fahrzeiten. Nicht-fahrplanbasierte Dienste, wie eine Taxifahrt, stehen zu jeder Zeit zur Verfügung. Deshalb wäre es nicht möglich, diese Zeiten statisch in der USDL-Beschreibung zu hinterlegen.

Insgesamt wurden basierend auf diesen Konzepten 12 verschiedene Transformationsregeln entwickelt. Für Adaption und Substitution existieren mehrere Regeln um Optimierungsziele abzudecken, etwa ob von der Ankunftszeit bevor-

zugt zurückgeplant wird oder ob die Abfahrtszeit maßgeblich ist. Weitere Regeln der Adaption können etwaige Wartezeiten minimieren. Pruning-Regeln dienen zum Entfernen von schlechten Alternativen. Da Services direkt Transformationsregeln für den Prozess mit einbringen können, sind außerdem Validierungsschritte notwendig, um die Terminierung des Transformationsprozesses und korrekte Prozesse sicherzustellen.

Alle Regeln liegen in einer Prioritätsreihenfolge vor. Pruning hat eine hohe Priorität, Einfügen aus dem Repository sollte dagegen nur durchgeführt werden, falls eine Transportlücke nicht anderweitig geschlossen werden kann. Der Algorithmus führt eine Reihe von Transformationsphasen bestehend aus Suche und Transformation durch. Jede Phase beginnt mit der Suche. Dabei wird immer mit der Regel höchster Priorität begonnen. Trifft die Bedingung der Regel (LHS) auf dem Graphen an keiner Stelle zu, wird mit der jeweils nächsten Regel fortgesetzt. Trifft keine Regel zu, endet der Algorithmus. Nach der Suche wird die Transformation der Regel auf alle Treffer angewendet und die aktuelle Phase endet [10].

Beispiel: Der Nutzer möchte von Darmstadt Mitte ab 07:00 Uhr nach Berlin Alexanderplatz (möglichst bis 11:30 Uhr) reisen. Der Planer konstruiert aus dieser Anfrage einen Graphen mit zwei attributierten Knoten: Start (07:00 Uhr, Darmstadt) und Ziel (11:30 Uhr, Berlin Alex.). Eine Kante zwischen beiden Knoten symbolisiert die Abfolge zwischen den Knoten, also den Reisewunsch, und somit die Aufgabestellung (Abbildung 1b, Schritt 0). Beide Knoten haben kein USDL-Attribut, daher scheiden Adaption und Substitution aus, es verbleibt das Einfügen. Entsprechend der Ortsangaben sowie des geringen Umfangs des Graphen liefert das Repository einen allgemeinen Dienst zurück, hier als Beispiel der Reise-Dienst der Lufthansa in Form einer USDL-Beschreibung. Daraus wird ein neuer Knoten (mit USDL-Attribut) erstellt und eingefügt (Schritt 1). Der neue Knoten besitzt noch keine Kontextattribute (Ort & Zeit), ist aber nach der USDL-Beschreibung substituierbar und wird daher in der nächsten Phase transformiert. Ein per USDL beschriebener Webservice des Reisedienstes wird mit den umrahmenden Kontextinformationen (von Start und Ziel) aufgerufen. Dieser wählt, hier anhand der Distanz, Fliegen als sinnvollste Reise-Modalität und liefert den Flugreise-Dienst zurück (Schritt 2). Da dem neuen Dienst ebenfalls noch Kontextinformationen fehlen, und kein Webservice zur Substitution enthalten ist, wird in Phase 3 eine Adaption durchgeführt und ein Webservice der Flugreise aufrufen. Der Service sucht nach entsprechenden Flughäfen und Flügen, hier Flug LH176 um 9:30 Uhr von Flughafen Frankfurt nach Berlin Tegel, und liefert diesen als Kontextinformationen zurück (Schritt 3). Hier wird auch deutlich, dass durch simultane Wahl von Orten (wie Flughäfen) und Zeiten Routing- sowie Scheduling kombiniert betrachtet werden. Der Dienst wählt, ähnlich der Verbindungssuche der Deutschen Bahn, Haltestellen, Verkehrsmittel sowie Abfahrtszeiten, um die Gesamtreisedauer zu minimieren, und nutzt dazu das umfangreiche domänenspezifische Wissen des Dienstleisters. Ein großer Teil der Transportlücke ist jetzt geschlossen. Es verbleiben kleinere Lücken, die in weiteren Phasen analog geschlossen werden. Natürlich können bei der Substi-

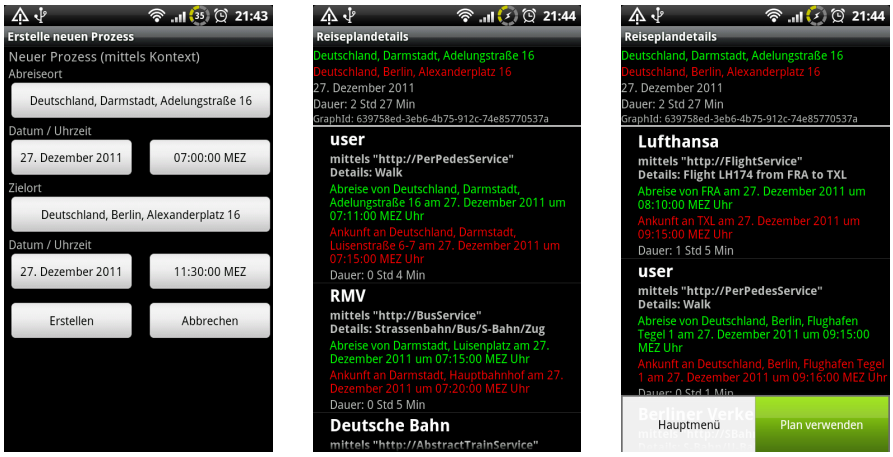
tution auch Graphen mit alternativen Dienstleistungen zurückgeliefert werden, etwa Flug- sowie Bahnreise als auch bereits mit Kontextinformationen (Flüge) versehene alternative Flugreise-Dienste. Mit der Adaption sind auch nachträgliche Änderungen möglich, beispielsweise ein späterer Flug aufgrund langer Anreise.

5 Implementierung

Im Rahmen der Arbeit wurde ein USDL-basiertes Service-Repository auf Basis von PostgreSQL und PostGIS entwickelt. Die Ortsinformationen werden aus den USDL-Beschreibungen extrahiert und können bei der Servicesuche verwendet werden. Der Zugriff auf das Repository erfolgt über SOAP/HTTP. Für ein realitätsnahes Szenario wurden eine Reihe von Diensten entwickelt, darunter ein Flug-Dienst auf Basis eines Crawlers für Lufthansa-Webseiten, Dummy-Services mit etlichen Haltestellen und zufälligen Verbindungen für die Deutsche Bahn, den RMV, die Berliner Verkehrsbetriebe, sowie ein Fußgängerservice.

Ein exemplarischer Client wurde als Android App realisiert (Abbildung 2). Eine Reise von Darmstadt nach Berlin wurde als Szenario zur Abdeckung der Dienste verwendet, hier kommen die Modalitäten zu Fuß, Bus, Zug, Flugzeug und S-Bahn kombiniert zum Einsatz. Die Kommunikation zwischen Planer und Client wurde mit der Kommunikations-Middleware MundoCore [3] implementiert.

Nach ersten Tests korreliert die Laufzeit einer Reiseplanung mit dem Umfang der Aufgabestellung. Der Haupteinflussfaktor sind die Zugriffe des Planers auf Webservices der Reisedienstleister, einschließlich einer Anfrage an das Repository sind dies maximal 3 Aufrufe für jeden Knoten. Das Beispielszenario umfasst final 8 echte Knoten, involviert 5 verschiedene Teilnehmer und wurde mit 17 Transformationen erstellt.



(a) Zieleingabe (b) Reiseplan (Teil 1) (c) Reiseplan (Teil 2)

Abbildung 2: Screenshots der Android App

6 Zusammenfassung

Eine Reiseplanung auf dieser Basis kann Dienstleister aktiv in die Planung mit einbeziehen, auf deren Fahrplanauskünfte und Buchungssysteme zurückgreifen und somit ideale, intermodale Reisepläne erstellen. Weiterhin wird im Rahmen der Verbreitung von Smartphones und Assistenzdiensten der Weg zu einer ineinandergreifenden Reiseunterstützung eröffnet.

Danksagung. Diese Arbeit wurde unterstützt durch das Theseus-Programm, gefördert durch das Bundesministerium für Wirtschaft und Technologie (Kennziffer: 01MQ07012).

Literatur

1. Aitenbichler, E.: Entwurf und Implementierung eines programmierten Graphersetzungssystems in Java. Master's thesis, Johannes Kepler Universität Linz (2000)
2. Aitenbichler, E., Borgert, S., Mühlhäuser, M.: Distributed Execution of S-BPM Business Processes. In: S-BPM ONE 2010 - The Subjectoriented BPM Conference. Springer (2011)
3. Aitenbichler, E., Kangasharju, J., Mühlhäuser, M.: MundoCore: A Light-weight Infrastructure for Pervasive Computing. Pervasive and Mobile Computing (2007)
4. Ambite, J., Barish, G., et al.: Getting from here to there: Interactive planning and agent execution for optimizing travel. In: Proc. of AAAI. pp. 862–869 (2002)
5. Arpinar, I.B., Zhang, R., Aleman-meza, B., Maduko, A.: Ontology-driven web services composition platform. In: Proc. of IEEE International Conference on e-Commerce Technology. pp. 6–9 (2004)
6. Benatallah, B., Sheng, Q.Z., Dumas, M.: The self-serv environment for web services composition. IEEE Internet Computing 7(1), 40–48 (January 2003)
7. BMWi: TEXO - Business Webs in the Internet of Services., http://www.theseus-programm.de/anwendungsszenarien/texo/default.aspx, Stand: 12.10.2010
8. Booth, J., Sistla, P., Wolfson, O., Cruz, I.: A data model for trip planning in multimodal transportation systems. In: Proc. of the EDBT. pp. 994–1005. ACM (2009)
9. Brennan, S., Meier, R.: STIS: Smart travel planning across multiple modes of transportation. In: Proc. of ITSC. pp. 666–671. IEEE (2007)
10. Daubert, J.: Service-Komposition von Reiseprozessen mittels Graphtransformation. Master's thesis, TU Darmstadt (2011)
11. Fleischmann, A.: Distributed Systems: Software Design and Implementation. Springer (1994)
12. Houda, M., Khemaja, M., Oliveira, K., Abed, M.: A public transportation ontology to support user travel planning. In: Proc. of RCIS. pp. 127–136. IEEE (2010)
13. Meier, R., Harrington, A., Cahill, V.: A framework for integrating existing and novel intelligent transportation systems. In: Proc. of ITSC. pp. 154–159. IEEE (2005)
14. Navabpour, S., Ghoraie, L., Malayeri, A., Chen, J., Lu, J.: An Intelligent Traveling Service Based on SOA. In: Proc. of Services. pp. 191–198. IEEE (2008)
15. SAP Research: USDL Specifications. http://www.internet-of-services.com/
16. Wang, J., Ding, Z., Jiang, C.: An Ontology-based Public Transport Query System. In: Proc. of Semantics, Knowledge and Grid (SKG). p. 62. IEEE (2007)