TripICS - a Web Service Composition System for Planning Trips and Travels (Extended Abstract)

Artur Niewiadomski[1], Wojciech Penczek[2]

[1] Institute of Computer Science, Siedlce University, Poland
`artur.niewiadomski@uph.edu.pl`
[2] Institute of Computer Science PAS, Warsaw and Siedlce University, Poland
`wpenczek@gmail.com`

1 Introduction

Automatic composition of Web services is a very active area of research which has provided a lot of important results [10, 1, 15, 6] as well as many implemented approaches [11, 12, 2, 3, 13]. In this paper we present the system TripICS - a real-life application of our Web service composition system PlanICS [8, 9, 13] to planning trips and travels around the world. While there are systems offering some support for planning excursions and travels [4, 5], our system uses advanced automated concrete planning methods [13, 16, 14]. TripICS is a specialization of the concrete planning viewed as a constrained optimization problem to the ontology containing services provided by hotels, airlines, railways, museums etc. The system finds an optimal plan (solution) satisfying the requirements of the user by applying the most efficient concrete planners of PlanICS based on a combination of an SMT-solver [7] with the nature inspired algorithm: GA, SA, and GEO [16, 14]. Contrary to PlanICS, the first phase of planning, called abstract planning, is realized by TripICS in a semi-automatic way by giving the user a possibility to choose the elements of an abstract plan using a Graphical User Interface (GUI). In the remainder we present: the description of the system TripICS, the theory behind it, and the implementation followed by some experimental results and conclusions.

2 TripICS Description

Our system is to allow the user for an easy and user-friendly planning of visits to interesting cities and places around the world in combination with travels in and out, arranged in the way satisfying the user's requirements. The general assumption is that the user would like to receive an optimal plan of a travel starting and ending in given locations and offering a possibility of visiting some specified cities within some specified dates. A plan is optimal if its quality value is the highest according to the given criteria. Below, we make the above description much more precise by giving three lists of requirements: 1) the user has to set (obligatory requirements), 2) the user can optionally set (optional requirements), and 3) the predefined quality requirements.

2.1 Obligatory Requirements

1. Trip starts and ends in two given locations (cities),
2. Trip starts from a given date (or a period of time) and lasts for a given number of days (optionally can be shorter or longer by a specified number of days),
3. Trip involves visiting given cities, each city within a specified minimal/maximal number of days,
4. Hotels with the free cancellation option are booked (optionally free cancellation is not required if this reduces the price by a given factor in $\%$),
5. In each city to be visited, the attractions specified in the optional rules, are available within the period of stay.

2.2 Optional Requirements

1. In each city, attractions (museums, exhibitions, matches, concerts, restaurants etc.) to attend are specified,
2. Quality of hotels is specified by giving a minimal number of stars (0 - 5) and a minimal score (0 - 10),
3. Travels do not last longer than a given number of hours.

Clearly, a plan should be optimal in the sense that the travels should conveniently fit to the stays and the prices should be as low as possible for a required quality of hotels. The aim of TripICS is to return such plans if they exist. Formally, these plans need to satisfy the user requirements as well as the quality requirements specified below.

2.3 Quality Requirements

1. A travel connection between two cities is always direct if it exists,
2. Costs, durations, and the numbers of breaks of the travels are minimized,
3. Costs of the visits are minimized while their standards and durations are maximized.

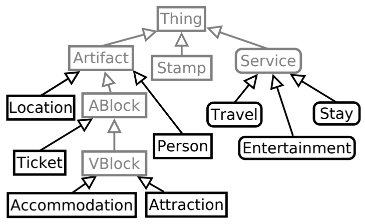
3 Theory behind TripICS

Typically, PlanICS realizes planning in three well defined phases called: abstract planning, offer collecting, and concrete planing, after receiving a user query specifying the requirements. In TripICS we depart from using an abstract planner, which does free the user from formulating a user query in the specification language. Instead, the user is given a possibility to set the obligatory and optional requirements about expected plans using GUI, described briefly in Section 3.2. All the user's choices, as well as the quality requirements, are automatically encoded as a user query and passed to the offer collector and the concrete planners. The available options result from the underlying ontology.

3.1 Ontology

This section discusses the ontology exploited by TripICS. Fig. 1 shows a part of the ontology corresponding to a travel domain. The ontology defines three service types $Travel$, $Stay$, and $Entertainment$ aimed at providing instances of the $Ticket$, $Attraction$, and $Accommodation$ object types (and operating also on objects of type $Person$ and $Location$) which are the trip elements constituting (among others) the abstract plan.

A ticket represents a journey from one location to another, for a certain price. An accommodation corresponds to a stay in some location, for a certain price as well. An attraction represents an admission ticket for an event, a reservation, or a confirmation that the attraction is available at the specified time. All these objects contain attributes describing contexts and details of the particular trip elements. We introduce also two auxiliary object types: $ABlock$ and $VBlock$. This is to avoid duplication of common attributes using the inheritance mechanism.



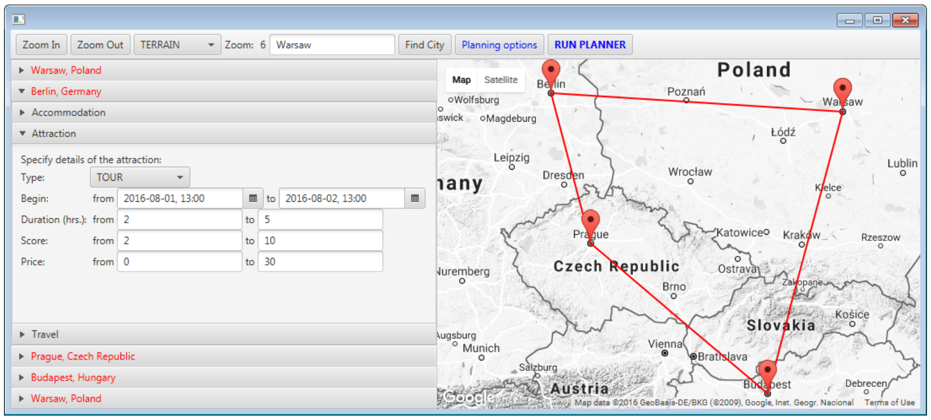
Object type	**Attributes' names and types**
$Location$	$continent$, $country$, $city$: $String$
$Person$	$name$: $String$; loc: $Location$
$ABlock$	$begin$, end: $Date$; $price$: $Real$; $type$: $String$
$Ticket$	$from, to$: $Location$; $breaks$: int
$VBlock$	$reviews$: $Real$; loc: $Location$
$Accommodation$	$stars$: Int; $freeCancel$: $Bool$
$Attraction$	$reserved$: $Bool$

Fig. 1. The TripICS ontology. The rectangles stand for object types while the rounded rectangles correspond to the service types. The types irrelevant for the working example are marked grey. The table describes the object types and their attributes.

For example, all the mentioned trip elements are described by the attributes $begin$, end, $price$, and $type$, and therefore they are inherited from the object type $ABlock$. The $type$ attribute defines the transportation type (bus, train, plane, ship, etc.), the accommodation type (hotel, guest house, hostel, apartment, etc.), or the attraction type (museum, exhibition, match, concert, restaurant etc.), when used in the $Ticket$, $Accommodation$, or $Attraction$ object, respectively. On the other hand, the number of breaks is specific to travels only, and thus the attribute $breaks$ is introduced in the object type $Ticket$. Each accommodation (and attraction) has been assigned a number stored in the attribute $reviews$ corresponding to the average score given by people who stayed there (or enjoyed the attraction) before. Similarly, since a fixed location is a common feature of accommodations and attractions, the attribute loc of type $Location$ is introduced in the class $VBlock$ and inherited by $Attraction$ and $Accommodation$ object types. The attributes are summarized in Fig. 1. Note that their meanings follow intuitively from their names.

3.2 Specifying requirements

First, using an intuitive GUI, the user inputs information about the dates and trip duration. The next step is to select the cities to be visited by clicking on the map or searching them by name. The cities are added to the list at the left hand side (see Fig. 2). Then, the user adds accommodations and attractions to enjoy in the particular cities, and inputs his preferences in the forms attached to the list.



Warsaw, Poland

Berlin, Germany

Accommodation

Attraction

Specify details of the attraction:

Type:	TOUR			
Begin:	from	2016-08-01, 13:00	to	2016-08-02, 13:00
Duration (hrs.):	from	2	to	5
Score:	from	2	to	10
Price:	from	0	to	30

Travel

Prague, Czech Republic

Budapest, Hungary

Warsaw, Poland

Fig. 2. TripICS GUI

Finally, the user starts the planning process using the dedicated button, and optionally sets some planner options, such as a planning algorithm (SMT, GA, IPH, SCGEO, SCSA[3] [16, 14]), timeout, maximal number of offers etc., as well as some parameters specific to the particular planning method, e.g., a population size of GA and IPH.

3.3 Collecting Offers and Planning

Basing on the city list and other data provided in the previous step, an abstract plan, i.e., a sequence of service types, is built. Next, this abstract plan is used by the *offer collector* (OC), i.e., the tool which in cooperation with the service registry queries real-world services. The service registry keeps an evidence of web services, registered accordingly to the service type system. Usually, each service type of the ontology represents a set of real-world services of similar functionality. For example, using the service type *Stay* one could register *Booking.com* as well as *Hilton* service.

OC queries web services of types present in the abstract plan and retrieves data called *offers*. An offer is a tuple of values representing a possible realization of one

[3] SMT - the SMT-based planner, GA- the GA-based planner, IPH - the initial population hybrid planner (SMT + GA), SCGEO - the SMT combined with GEO planner, SCSA - the SMT combined with SA planner, where SMT (Satisfiability Modulo Theories), GA (Genetic Algorithm), SA (Simulated Annealing), GEO (Generalised Extremal Optimization)

service type of the plan. Each value corresponds to an attribute of some object processed by the service type. The offers collected from a single service type of the plan constitute so called *offer sets*. The offers are searched by a *concrete planner* in order to find the best solution satisfying all constraints and maximizing the quality function. Thus, the concrete planning problem can be formulated as a constrained optimization problem (see [13]). Its solution consists in selecting one offer from each offer set such that all constraints are satisfied and the value of the quality function is maximized.

The constraints and the quality function result from the user requirements and preferences what is shown in the next subsection.

3.4 Constraints and Quality Function

The constraints and the quality function play a crucial role in the planning process. In this section, using a simple example, we show how the user requirements and preferences in combination with several general rules (described in Sec. 2) result in a set of constraints and a quality function.

Example 1. Assume that the user wants to make a trip on the 15th of August from Warsaw (W) to Berlin (B) and then back in a few days. In Berlin, he prefers to stay in a 3-star hotel for 3 days and during the visit he plans to take a city tour and attend a concert. The specified requirements result in an abstract plan consisting of the following 5 service types: ($Travel$, $Stay$, $Entertainment$, $Entertainment$, $Travel$). Then, OC searches for the matching offers, and retrieves the following example five offer sets ($O^1, \dots, O^5$).

$O^1(Travel)$

id	begin	end	price	type	from	to	breaks
1	15.08, 10:40	15.08, 12:05	565	plane	W	B	0
2	15.08, 06:20	15.08, 07:45	565	plane	W	B	0
3	15.08, 07:20	15.08, 12:05	533	plane	W	B	1
4	15.08, 14:05	15.08, 19:18	170	train	W	B	0
5	15.08, 18:05	15.08, 22:58	276	train	W	B	0

$O^2(Stay)$

id	begin	end	price	type	score	loc	stars	freeCanc
1	15.08, 14	18.08, 11	1044	hotel	9.1	B	3	yes
2	15.08, 15	18.08, 11	1211	hotel	9.1	B	3	yes
3	15.08, 15	18.08, 12	1729	hotel	9.0	B	3	yes
4	15.08, 15	18.08, 11	1032	hotel	7.0	B	3	yes
5	15.08, 15	18.08, 12	1259	hotel	8.9	B	3	yes

$O^3(Entertainment)$

id	begin	end	price	type	score	loc	reserv.
1	15.08, 18	15.08, 21	84	tour	8.5	B	yes
2	16.08, 12	16.08, 15	84	tour	8.5	B	yes
3	16.08, 15	16.08, 18	84	tour	8.5	B	yes
4	17.08, 12	17.08, 15	79	tour	7.2	B	yes
5	17.08, 15	17.08, 18	79	tour	7.2	B	yes

$O^4(Entertainment)$

id	begin	end	price	type	score	loc	reserv.
1	16.08, 20	16.08, 23	280	concert	7.2	B	yes
2	16.08, 21	17.08, 1	130	concert	8.1	B	yes
3	17.08, 21	18.08, 1	110	concert	3.0	B	yes
4	17.08, 18	17.08, 22	580	concert	9.3	B	yes
5	16.08, 20	16.08, 23	164	concert	7.9	B	yes

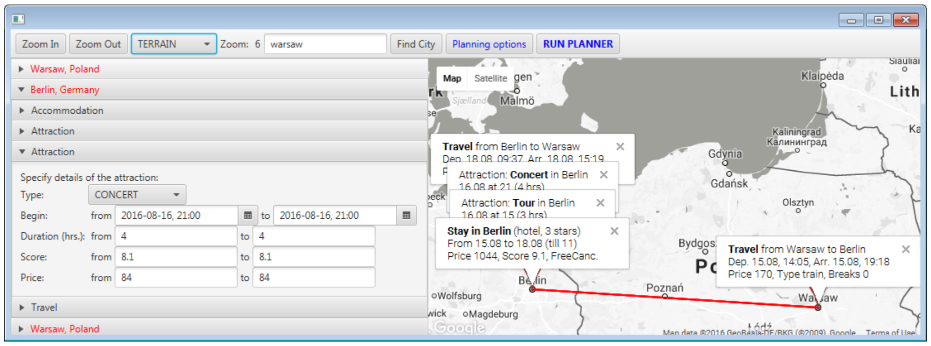
$O^5(Travel)$

id	begin	end	price	type	from	to	breaks
1	18.08, 11:50	18.08, 16:30	429	plane	B	W	1
2	18.08, 15:10	18.08, 19:30	524	plane	B	W	1
3	18.08, 08:50	18.08, 10:10	561	plane	B	W	0
4	18.08, 09:37	18.08, 15:19	170	train	B	W	0
5	18.08, 14:37	18.08, 20:36	276	train	B	W	0

This example deals with a plan of length 5 where every service of the plan has 5 possible realizations. Thus, the search space is of size $5^5 = 3125$ as there is so many possible offer combinations. However, the number of plans (solutions) is much lower if we take constraints into account.

For example, assume that the user wants to synchronise travels and hotel in such a way that the time between arrival and hotel check-in is not longer than 3 hours. Similarly, the return travel should be not later than 3 hours after the hotel check-out time. After adding these two constraints the number of the possible solutions decreases to 2200. Another constraint could be to have at least a three-hour break between attractions. When this constraint is taken into account, there are only 1408 possible solutions. The underlying constraints are encoded by the following expressions: $(o_2.begin - o_1.end \leq 3)$, $(o_5.begin - o_2.end \leq 3)$, $(o_4.begin - o_3.end \geq 3)$, where o_i represents an offer from the i-th offer set.

As to the quality function, the user can choose between several schemes, but he can also enable/disable some of the function components. For example, if the user prefers only to minimize the total price, the quality function is expressed by $W_{price} * \sum_{i=1..5} o_i.price$, and the optimal solution is $(4, 4, 5, 3, 4)$ with the price 1561, where W_{price} is some negative constant (a weight). The numbers in the sequence correspond to the numbers of the offers in the corresponding offer sets. That is, both the travels are by train for the price of 170 each, the stay is in the cheapest hotel, and the cheapest tour and concert are chosen. However, if the user also wants to maximize the reviews of the stay and attractions, the quality function is then as follows: $W_{price} * \sum_{i=1..5} o_i.price + W_{score} * \sum_{i=2..4} o_i.score$. Assuming $W_{price} = -1$ and $W_{score} = 10$, we obtain the optimal solution $(4, 1, 3, 2, 4)$ where the accommodation and attractions with a low price and a high score are chosen. Fig. 3 presents the resulting plan.



Zoom: 6 warsaw

Specify details of the attraction:

Type: CONCERT

Begin: from 2016-08-16, 21:00 to 2016-08-16, 21:00

Duration (hrs.): from 4 to 4

Score: from 8.1 to 8.1

Price: from 84 to 84

Travel from Berlin to Warsaw
Dep. 18.08, 09:37, Arr. 18.08, 15:19

Attraction: **Concert** in Berlin
16.08 at 21 (4 hrs)

Attraction: **Tour** in Berlin
16.08 at 15 (3 hrs)

Stay in Berlin (hotel, 3 stars)
From 15.08 to 18.08 (till 11)
Price 1044, Score 9.1, FreeCanc.

Travel from Warsaw to Berlin
Dep. 15.08, 14:05, Arr. 15.08, 19:18
Price 170, Type train, Breaks 0

Fig. 3. The example plan

4 Tripics Implementation and Experiments

The Tripics application is implemented in Java. It consists of several planning engines, Offer Collector, and the GUI module. GUI exploits GMapsFX [17] project which provides a wrapper to the Google Map's Javascript API, allowing to exploit and interact with maps using a pure Java. The map is the central component in GUI. It is used to

Table 1. The experimental results for the travel benchmarks. In each entry of the table having three rows of values, the first row contains the average values (bold), the second row contains the best values (normal font), the third row contains the standard deviation (italic).

	SMT	**SCSA**	**SCGEO**	**GA**		**IPH$_1$**	**IPH$_{500}$**
Instance	**t[s] / Q**	**t[s] / Q**	**t[s] / Q**	**t[s] / Q**	**P [%]**	**t[s] / Q**	**t[s] / Q**
T1	**52.4 / 228.3**	**0.9 / 222.0**	**0.8 / 221.4**	**1.7 / 201.5**	80	**1.8 / 202.5**	**1.9 / 222.8**
	47 / 228.3	0.8 / 222.0	0.7 / 222.0	1.5 / 222.0		1.6 / 222.0	1.6 / 228.3
	3.6 / 0	*0.1 / 0.0*	*0.0 / 4.1*	*0.1 / 13*		*0.1 / 18.5*	*0.1 / 6.3*
T2	**400.6 276.2**	**1.2 / 268.9**	**1.0 / 268.4**	**1.8 / 222.7**	92	**1.9 / 223.6**	**1.7 / 277.1**
	400.3 / 276.2	1.0 / 276.2	0.9 / 276.2	1.7 / 284.2		1.8 / 284.2	1.6 / 284.2
	0.3 / 0	*0.1 / 13.8*	*0.1 / 10.9*	*0 / 37.3*		*0.1 / 38.5*	*0.1 / 7.6*
T3	**400.3 / 79.3**	**1.5 / 270.3**	**1.2 / 249.7**	**1.8 / 235.7**	72	**2.1 / 209.7**	**2.8 / 266.2**
	400.3 / 79.3	1.3 / 290.4	1.0 / 290.4	1.6 / 306.1		2 / 269.3	2.5 / 290.3
	0 / 0	*0.1 / 16.8*	*0.1 / 26.0*	*0.1 / 33.8*		*0.1 / 39.6*	*0.2 / 4.9*
T4	**400.2 / 106.9**	**1.7 / 217.3**	**1.8 / 216.7**	**3.3 / 178.5**	70	**3.2 / 169**	**3.4 / 173.6**
	400.2 / 106.9	1.6 / 217.3	1.7 / 217.3	3.1 / 238.8		3.1 / 214.8	2.7 / 210.4
	0 / 0	*0.1 / 0.0*	*0.1 / 4.1*	*0.1 / 27.4*		*0.1 / 22.5*	*0.4 / 10.5*
T5	**400.3 / 73.4**	**2.1 / 191.5**	**2.2 / 190.8**	**3.2 / 211.2**	76	**3.5 / 193.8**	**7.1 / 331.3**
	400.3 / 73.4	1.9 / 191.7	2.0 / 191.7	3 / 281.4		3.4 / 336.9	3.7 / 344.8
	0 / 0	*0.1 / 0.5*	*0.1 / 1.6*	*0.1 / 33.7*		*0.1 / 39.3*	*1.9 / 9.2*
T6	**400.4 / 12.5**	**2.5 / 277.6**	**2.7 / 262.0**	**3.2 / 206.9**	80	**3.8 / 199.7**	**8 / 304.1**
	400.4 / 12.5	2.3 / 320.3	2.4 / 320.3	3.1 / 286.1		3.7 / 264.4	7.8 / 308.5
	0 / 0	*0.1 / 19.6*	*0.2 / 16.4*	*0.1 / 36.7*		*0.1 / 31.5*	*0.2 / 3.1*
T7	**400.3 / 19.6**	**2.6 / 210.9**	**3.6 / 210.1**	**4.7 / 158.8**	80	**5.6 / 186.2**	**4.3 / 181.2**
	400.3 / 19.6	2.5 / 214.2	3.4 / 214.2	4.6 / 258.6		5.1 / 211.1	4.1 / 191.9
	0 / 0	*0.1 / 5.8*	*0.1 / 6.2*	*0.1 / 38.1*		*0.2 / 14.3*	*0.1 / 5.8*
T8	**400.4 / 38.1**	**3.0 / 162.0**	**4.0 / 155.4**	**4.6 / 177.3**	60	**6.1 / 95.1**	**9.3 / 226.4**
	400.3 / 38.1	2.8 / 180.4	3.8 / 180.4	4.3 / 237.0		5.7 / 143.2	9.1 / 242.3
	0.1 0	*0.1 18.6*	*0.1 20.9*	*0.1 38.8*		*0.2 23.5*	*0.1 11.2*
T9	**400.5 / 35**	**3.6 / 258.5**	**4.5 / 255.0**	**4.5 / 180.2**	62	**6.2 / 187,4**	**10.8 / 187.1**
	400.4 / 35	3.4 / 266.2	4.2 / 266.2	4.4 / 285.6		5.4 / 250.3	10.4 / 227.7
	0.1 / 0	*0.1 / 7.6*	*0.1 / 10.6*	*0 / 38.2*		*0.6 / 33*	*0.2 / 19.9*

specify user requirements as well as to visualize plans. The application is still being extended and improved.

We have evaluated the efficiency of TripICS focusing on the planning modules. At the moment our Offer Collector supports only a limited number of services. Thus, we have used several benchmarks generated by our software Offer Generator. They have been scaled by the length of a plan and the number of offers. Plans of length 5 have been found for the benchmarks T1, T2, and T3, of length 9 for T4, T5, and T6, and of length 13 for the remaining examples. The number of offers equals $2^8 = 256$ for the benchmarks T1, T4, and T7; $2^9 = 512$ for T2, T5, and T8; and $2^{10} = 1024$ for the other cases. This gives a number of the potential solutions varying from $256^5 = 2^{40}$ for T1 to $1024^{13} = 2^{130}$ for T9. The tested examples involved from 13 to 43 constraints. We have compared several planning algorithms taking into account the computation time and the quality of the solutions found. The methods combining SMT with nature-inspired algorithms appear to be the most efficient. They are able to find solutions of very high quality within a few seconds only, which makes them acceptable for the user. The detailed results are given in Table 1 and summarised in Fig. 4.

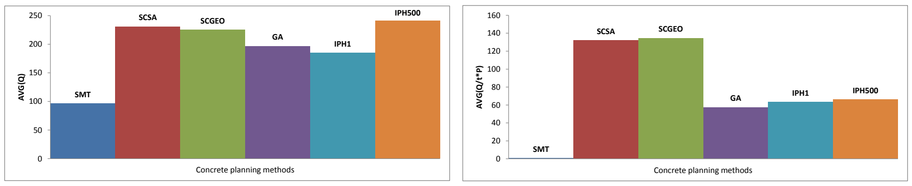


Fig. 4. A comparison of the average quality of solutions produced by the concrete planning methods (left) and average efficiency comparison of the concrete planning methods (right). By efficiency we mean $quality/time * probability$.

5 Conclusions

We have presented a preliminary version of our system TripICS, which can be used for planning trips and travels around the world. Our motivation for developing this system was twofold. Firstly, we wanted to show that web service composition can be successfully used in practice for real-life applications, and secondly our aim is to offer a new useful tool, which could be publicly used.

References

1. S. Ambroszkiewicz. Entish: A language for describing data processing in open distributed systems. *Fundam. Inform.*, 60(1-4):41–66, 2004.
2. D. Berardi, F. Cheikh, G. De Giacomo, and F. Patrizi. Automatic service composition via simulation. *Int. J. Found. Comput. Sci.*, 19(2):429–451, 2008.
3. D. Calvanese, G. De Giacomo, M. Lenzerini, M. Mecella, and F Patrizi. Automatic service composition and synthesis: the roman model. *IEEE Data Eng. Bull.*, 31(3):18–22, 2008.
4. D. Damljanovic and V. Devedzic. Applying semantic web to e-tourism. In *In Ma, Z.,Wang, H. (Eds.), The Semantic Web for Knowledge and Data Management: Technologies and Practices*, pages 243–265. IGI Global, New York, 2008.
5. D. Damljanovic and V. Devedzic. Applying semantic web to e-tourism. In *In Mehdi Khosrow-Pour (Ed.), Encyclopedia of Information Science and Technology*, pages 3426–3432. IGI Global, Hershey, 2009.
6. G. De Giacomo, M. Mecella, and F. Patrizi. Automated service composition based on behaviors: The roman model. In Athman Bouguettaya, Quan Z. Sheng, and Florian Daniel, editors, *Web Services Foundations*, pages 189–214. Springer, 2014.
7. L. De Moura and N. Bjorner. Satisfiability modulo theories: Introduction and applications. *Commun. ACM*, 54(9):69–77, September 2011.

8. D. Doliwa, W. Horzelski, M. Jarocki, A. Niewiadomski, W. Penczek, A. Półrola, and J. Skaruz. HarmonICS - a tool for composing medical services. In *ZEUS*, pages 25–33, 2012.
9. D. Doliwa, W. Horzelski, M. Jarocki, A. Niewiadomski, W. Penczek, A. Pólrola, M. Szreter, and A. Zbrzezny. Planics - a web service composition toolset. *Fundam. Inform.*, 112(1):47–71, 2011.
10. T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.
11. Z. Li, L. O'Brien, J. Keung, and X. Xu. Effort-oriented classification matrix of web service composition. In *Proc. of the Fifth International Conference on Internet and Web Applications and Services*, pages 357–362, 2010.
12. W. Nam, H. Kil, and D. Lee. Type-aware web service composition using boolean satisfiability solver. In *Proc. of CEC'08 and EEE'08*, pages 331–334, 2008.
13. A. Niewiadomski, W. Penczek, J. Skaruz, M. Szreter, and M. Jarocki. SMT versus Genetic and OpenOpt Algorithms: Concrete Planning in the PlanICS Framework. *Fundamenta Informaticae*, 135(4):451–466, 2014.
14. A. Niewiadomski, J. Skaruz, P. Switalski, and W. Penczek. Concrete Planning in PlanICS Framework by Combining SMT with GEO and Simulated Annealing. *To appear in Fundamenta Informaticae*, 2016.
15. J. Rao and X. Su. A survey of automated web service composition methods. In *Proc. of SWSWPC'04*, volume 3387 of *LNCS*, pages 43–54. Springer, 2005.
16. J. Skaruz, A. Niewiadomski, and W. Penczek. Hybrid Planning by Combining SMT and Simulated Annealing. In Z. Suraj and L. Czaja, editors, *Proceedings of the 24th International Workshop on Concurrency, Specification and Programming, Rzeszow, Poland, September 28-30, 2015.*, volume 1492 of *CEUR Workshop Proceedings*, pages 173–176. CEUR-WS.org, 2015.
17. R. Terpilowski. GMapsFX - a JavaFX API for Google Maps, 2016.