# Cost-minimal adapters for services

Jan Sürmeli

Humboldt-Universität zu Berlin, Institut für Informatik,
Unter den Linden 6, 10099 Berlin
`suermeli@informatik.hu-berlin.de`

**Abstract** The composition of compatible services is central in service orientation. Adapters resolve incompatibilities between services. Adapter synthesis generates an adapter $A$ for two given services $N_1$ and $N_2$. Generally, there exist several different adapters for $N_1$ and $N_2$. In this paper, we suggest a framework to express preference between those adapters. Additionally, we sketch the synthesis of a cost-minimal adapter.

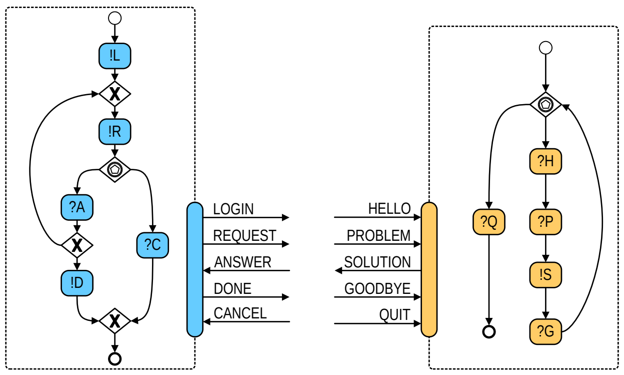
## 1 Introduction

We understand a *service* [6] as a component with an *inner process* and an *interface* for message exchange with other services. The actions of the inner process may be linked with the interface, declaring which actions *receive* and *send* which type of messages. A central concept of service orientation is the *composition* of services. Clearly, it is only feasible to compose *compatible* services. There exist four core aspects [5]: Syntactical, behavioral, semantical, and non-functional. In this paper, we concentrate on *weak termination*, a behavioral compatibility notion, similar to soundness [10] in business processes. A set of services is compatible w.r.t. weak termination, iff the composition of its elements is free of deadlocks and livelocks. An *adapter* [12] solves the problem that two services $N_1$ and $N_2$ are incompatible by mediating between them. An adapter $A$ is a service, such that the set $N_1$, $N_2$, and $A$ of services is compatible. The idea of *adapter synthesis* is to automatically generate an adapter for two services. Generally, there are different adapters for two services $N_1$ and $N_2$. In this paper, we propose a framework to express *preference* between such adapters by means of *cost models* and *cost functions*. We discuss two cost models. For one cost model, we sketch the synthesis of a *cost-minimal adapter*.

## 2 Running example

As a running example, we introduce two simple incompatible services, which are depicted in Fig. 1 in a notion similar to BPMN: Boxes are tasks, diamonds are splits and merges. Initialization and termination are represented by circles and bold circles, respectively. The dashed line encapsulates a service. A rounded box on the dashed line is a port, consisting of input and output message types. We further explain syntax and semantics by describing the actual models.

Service $N_1$ has one port with the input message types ANSWER and CANCEL, and the output message types LOGIN, REQUEST, and DONE. Initially $N_1$ sends a LOGIN message. This is modeled as a task named !L, where ! stands for *sending*, and L stands for LOGIN – we abbreviate the names of the message types. Then, $N_1$ enters a loop. In each iteration, it sends an R (resembled by !R) and waits for an A or a C. If it receives a C, the loop is left. If it receives an A, it decides internally between starting another iteration, or leaving the loop. In the latter case, it sends a D to inform its environment that it is done sending requests. Upon leaving the loop, it terminates. Service $N_2$ serves its environment by receiving a H followed by a P. It then returns a S followed by receiving a G. Initially and after receiving a G, $N_2$ can receive a Q to terminate.



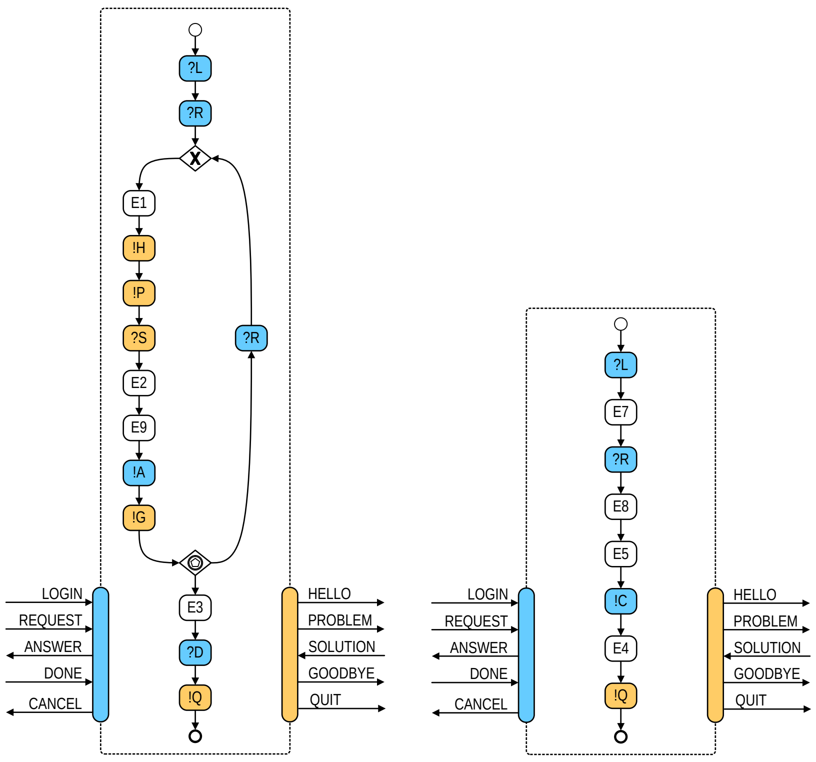
(a) $N_1$ (b) $N_2$

Figure 1: Running example: Two incompatible services $N_1$ and $N_2$

Obviously, $N_1$ and $N_2$ are incompatible. Even if one tries to map the interfaces as far as possible (i.e. $L \mapsto H$, $R \mapsto P$, $S \mapsto A$, $D \mapsto G$), the composition deadlocks: After one iteration of the loop, $N_2$ waits for another H, and $N_1$ waits for an A.

There exist different adapters for $N_1$ and $N_2$. Figure 2 depicts two adapters $A_1$ and $A_2$ for our running example. Adapter $A_1$ sends the missing H in each loop iteration. Once $N_1$ decides to be done, it quits $N_2$. In contrast to the previously studied models, $A_1$ executes tasks neither starting with ? nor !, e.g. $E_1$. Such a task is internal, that is, neither sending nor receiving a message. The label $E_1$ refers to a message transformation in Fig. 3(a). We explain this in more detail in the next section. Adapter $A_2$ resolves the incompatibility trivially by sending a C to $N_1$, and a Q to $N_2$.

One might *prefer* one adapter over the other. For instance, $A_1$ could be preferable because it enforces both services to enter their main part. In contrast to that, $A_2$ simply quits both services, which does not seem very useful. However, one could also prefer $A_2$ over $A_1$, because executing the main part of the services requires many costly message transformations.



(a) $\mathsf{A}_1$ (b) $\mathsf{A}_2$

Figure 2: Running example: Two adapters $\mathsf{A}_1$ and $\mathsf{A}_2$

## 3 General synthesis approach

In this section, we propose a general approach to synthesize a cost-minimal adapter. We first explain the approach in [2] to synthesize an arbitrary adapter. Second, we explain how to extend this approach such that it yields a cost-minimal adapter.

We synthesize an adapter for $N_1$ and $N_2$ based on a given *specification of elementary activities* (SEA). Such an SEA contains all semantical activities an adapter may perform. An elementary activity is a rule of the form $x_1, \ldots, x_m \mapsto y_1, \ldots, y_n$, where $x_1, \ldots, x_m, y_1, \ldots, y_n$ are message types. For each message type $x_i$, a message of that type is consumed, for each message type $y_j$, a message of this type is produced. Message transformations are performed on internal buffers. That is, the adapter stores incoming and intermediate messages locally.

Continuing the running example, Fig. 3(a) shows an SEA $\{\mathsf{E}_1, \ldots, \mathsf{E}_9\}$. We recognize all but one message type from the running example: Message type $\mathsf{X}$ is a temporary message to remember that rule $\mathsf{E}_2$ has been applied. That is, that at least one $\mathsf{S}$ has been translated to an $\mathsf{A}$. Using such intermediate messages is a mechanism to cope with dependencies between rules. We identify which adapter uses which rules. Adapter $\mathsf{A}_1$ (Fig. 2(a)) executes $\mathsf{E}_1$, $\mathsf{E}_2$, $\mathsf{E}_9$, and $\mathsf{E}_3$. Adapter $\mathsf{A}_2$ (Fig. 2(b)) executes $\mathsf{E}_7$, $\mathsf{E}_8$, $\mathsf{E}_5$, and $\mathsf{E}_4$. Finally, the adapter $\mathsf{A}_{\min}$ (Fig. 3(b)) executes $\mathsf{E}_1$, $\mathsf{E}_2$, $\mathsf{E}_3$, $\mathsf{E}_9$, $\mathsf{E}_8$, $\mathsf{E}_6$, and $\mathsf{E}_4$.

One advantage of this approach is that adapter synthesis can be reduced to *partner synthesis*. Intuitively, partner synthesis [11] solves the problem to find a compatible service $N_2$ for a given service $N_1$, called partner of $N_1$. First, we create an *engine* $E$ from the SEA. The engine has three ports, one for each $N_1$ and $N_2$, and one for a *control service*. We compose $N_1$, $E$, and $N_2$. We synthesize a control service $C$. The composition of $E$ and $C$ serves as an adapter for $N_1$ and $N_2$. We shortly discuss the limitations of this approach. It is possible to adapt more than just two services at once, but only by a central adapter. Additionally, this approach is adapts single instances of each service. If one desires to adapt $n$ instances of one service, it is required to treat them each as a different service, instead. This is obviously infeasible if the number of instances is variable and not known beforehand.

We propose to specify costs based on the SEA, because it contains all activities which are known before synthesis. We then follow the above approach and reduce the problem to synthesize a cost-minimal adapter to the problem to synthesize a cost-minimal partner.

## 4 Cost models and cost functions

In this section, we introduce two formal constructs: *Cost models* and *cost functions*. A cost model determines how costs are represented, aggregated, and compared. A cost function specifies the costs for executing a rule of the SEA $\Sigma$. These cost may resemble monetary costs, for calling a web service, or penalties. Given a cost model and a cost function, the costs of an adapter are well-defined.

Formally, a cost model $\mathcal{C} = [\mathcal{D}, \mathcal{S}, \leq]$ consists of a set $\mathcal{D}$, called *domain* of $\mathcal{C}$, a function $\mathcal{S} : \mathcal{D}^* \to \mathcal{D}$, called *sequence aggregator function* (SAF) of $\mathcal{C}$, and a partial order $\leq$ on $2^{\mathcal{D}}$, called *set ordering relation* (SOR) of $\mathcal{C}$.

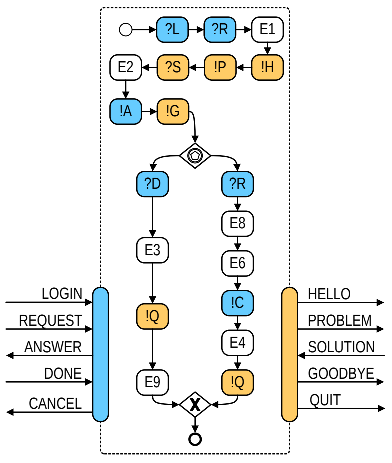
A *cost function* $\mathcal{F} : \Sigma \to \mathcal{D}$ specifies the cost of executing a single rule $a \in \Sigma$. We combine a cost function with a cost model to determine the costs of a sequence of rules. Let $\sigma = a_1 \dots a_n \in \Sigma^*$. We define the costs of $\sigma$ as $\langle \mathcal{F}, \mathcal{C} \rangle(\sigma) := \mathcal{S}(\mathcal{F}(a_1) \dots \mathcal{F}(a_n))$. Let $L, L' \subseteq \Sigma^*$ be sets of sequences. We define $\langle \mathcal{F}, \mathcal{C} \rangle(L) := \{\langle \mathcal{F}, \mathcal{C} \rangle(\sigma) \mid \sigma \in L\}$. We define $L \leq^{\mathcal{F}} L'$ iff $\langle \mathcal{F}, \mathcal{C} \rangle(L) \leq \langle \mathcal{F}, \mathcal{C} \rangle(L')$.

Let $N_1$ and $N_2$ be incompatible services. The *costs of an adapter* $A$ *w.r.t.* $\mathcal{F}$*,* $\mathcal{C}$*,* $N_1$*, and* $N_2$ are then determined by inspecting the *terminating runs* of the composition of $N_1$, $N_2$, and $A$. A terminating run results in a common final state of all services. We define $\langle \mathcal{F}, \mathcal{C}, N, Q \rangle(A) := \{\langle \mathcal{F}, \mathcal{C} \rangle(\sigma|_\Sigma) \mid \sigma \text{ is a terminating run of } N \oplus A \oplus Q\}$, where $\sigma|_\Sigma$ denotes the restriction of $\sigma$ to alphabet $\Sigma$. For two adapters $A, A'$, we define $A \leq^{\mathcal{F},N,Q} A'$ if and only if $\langle \mathcal{F}, \mathcal{C}, N, Q \rangle(A) \leq^{\mathcal{F}} \langle \mathcal{F}, \mathcal{C}, N, Q \rangle(A')$.

A general limitation of this approach is the fact that costs for a sequence of rule executions are built from the costs of the executions of single rules. One may desire

| Rule name | Rule body | Costs |
|---|---|---|
| $\mathsf{E}_1$ | $\mathsf{L}, \mathsf{R} \mapsto \mathsf{H}, \mathsf{P}, \mathsf{L}$ | 5 |
| $\mathsf{E}_2$ | $\mathsf{S} \mapsto \mathsf{A}, \mathsf{G}, \mathsf{X}$ | 5 |
| $\mathsf{E}_3$ | $\mathsf{D} \mapsto \mathsf{Q}$ | 0 |
| $\mathsf{E}_4$ | $\mapsto \mathsf{Q}$ | 0 |
| $\mathsf{E}_5$ | $\mapsto \mathsf{C}$ | 30 |
| $\mathsf{E}_6$ | $\mathsf{X} \mapsto \mathsf{C}$ | 10 |
| $\mathsf{E}_7$ | $\mathsf{L} \mapsto$ | 0 |
| $\mathsf{E}_8$ | $\mathsf{R} \mapsto$ | 0 |
| $\mathsf{E}_9$ | $\mathsf{X} \mapsto$ | 0 |

(a) Cost function $\mathsf{F}$



(b) Cost-minimal adapter $\mathsf{A}_{\mathrm{min}}$

Figure 3: A cost function $\mathsf{F}$, and a cost-minimal adapter $\mathsf{A}_{\mathrm{min}}$ w.r.t. cost function $\mathsf{F}$, cost model $\mathcal{T}$, and services $\mathsf{N}_1$ and $\mathsf{N}_2$

to express dependencies, for instance, executing rule $E = x_1, \ldots, x_m \mapsto y_1, \ldots, y_n$ could be cheaper if a message $X$ was received beforehand. This can be partly realized by intermediate messages. For this case, one would introduce three new rules $R = X \mapsto X, \mathit{received}_X$, $R' = \mathit{received}_X \mapsto$ , and $E' = \mathit{received}_X, x_1, \ldots, x_m \mapsto \mathit{received}_X, y_1, \ldots, y_n$. One would apply the appropriate lower costs to $E'$. An open question is to find the class of dependencies one can express in this way. For example, it is not possible to declare that the costs for executing a rule are reduced by factor two each time.

In the remainder of this section, we discuss two cost models, and apply these cost models to our running example. For that purpose, we define a cost function $\mathsf{F}$ based on the SEA in Fig. 3(a). Most of the rules have costs of zero. However, $\mathsf{E}_1$ and $\mathsf{E}_1$ have costs of 5, because the message content has to be translated to a different format. Rule $\mathsf{E}_5$ has a penalty of 30, to symbolize that we do not prefer to send a $\mathsf{C}$ to $\mathsf{N}_1$. Rule $\mathsf{E}_6$ has a lower penalty, because it may only be applied after at least one request has been answered.

### 4.1 A cost model for worst case total costs

The idea of this cost model is to compute the total costs of each run, and select the supremum to compare two adapters. Total costs are determined by addition. We prefer an adapter $A_1$ over an adapter $A_2$ if the worst-case total costs of $A_1$ are lower than the worst case total costs of $A_2$. In order to enable analysis, we choose the natural numbers as domain. The advantage is an inherent monotony. We define the cost model $\mathcal{T}$ as the cost model with domain $\mathcal{D}_\mathcal{T} := \mathbb{N}_0$, the SAF $\mathcal{S}_\mathcal{T}(a_1 \ldots a_n) := a_1 + \ldots + a_n$, and SOR $X \leq_\mathcal{T} Y$ if and only if $\sup(X) \leq \sup(Y)$. Thereby, $\mathbb{N}_0$ denotes the set of natural numbers, $+$ denotes integer addition,

$\sup(X) \in \mathbb{N}_0 \cup \{\infty\}$ denotes the supremum (least upper bound) of $X$, and $\leq$ denotes the natural order on $\mathbb{N}_0 \cup \{\infty\}$.

Comparing the adapters of our running example, we find: $\mathsf{A}_1 \mapsto \{10, 20, 30, \ldots\}$, $\mathsf{A}_2 \mapsto \{30\}$, $\mathsf{A}_{\min} \mapsto \{10, 20\}$. We prefer $\mathsf{A}_{\min}$, because $20 \leq 30 \leq \infty$. We believe that it is obvious that $\mathsf{A}_{\min}$ is the cost-minimal adapter w.r.t. $\mathsf{F}$, $\mathcal{T}$, $\mathsf{N}_1$ and $\mathsf{N}_2$. It is impossible to adapt $\mathsf{N}_1$ and $\mathsf{N}_2$ with less costs, because $\mathsf{N}_1$ decides whether it sends another request, or whether it is done.

### 4.2 A cost model for worst case average costs

The disadvantage of cost model $\mathcal{T}$ is that all adapers with unbounded worst case total costs are equivalent. That is, we may not distinguish between them. However, there might be services $N_1$ and $N_2$ which are not adaptable by a service with bounded worst case costs. In this case, we would like to be able to distinguish two adapters $A_1$, $A_2$ with unbounded costs. Intuitively, we compare the costs for executing an additional rule in each adapter. We evaluate the average costs of each run (instead of the total costs), and use again the supremum to compare. We prefer an adapter $A_1$ over an adapter $A_2$ if the worst-case average costs of $A_1$ are lower than the worst case average costs of $A_2$. Obviously, this cannot be done in the natural numbers anymore. Hence, we select the non-negative rational numbers as domain. We define the cost model $\mathcal{A}$ as the cost model with domain $\mathcal{D}_{\mathcal{A}} := \mathbb{Q}_0^+$, the SAF $\mathcal{S}_{\mathcal{A}}(a_1 \ldots a_n) := \frac{a_1 + \ldots + a_n}{n}$, and SOR $X \leq_{\mathcal{A}} Y$ if and only if $\sup(X) \leq \sup(Y)$. Thereby, $\mathbb{Q}_0^+$ denotes the set of non-negative rational numbers, $+$ denotes rational number addition, $\sup(X) \in \mathbb{Q}_0^+ \cup \{\infty\}$ denotes the supremum (least upper bound) of $X$, and $\leq$ denotes the natural order on $\mathbb{Q}_0^+ \cup \{\infty\}$.

Comparing the adapters of our running example, we find: $\mathsf{A}_1 \mapsto \{\frac{10}{4}, \frac{20}{7}, \frac{30}{10}, \ldots\}$, $\mathsf{A}_2 \mapsto \{\frac{30}{4}\}$, $\mathsf{A}_{\min} \mapsto \{\frac{10}{4}, \frac{20}{5}\}$. We prefer $\mathsf{A}_1$, because $3 \leq 4 \leq 7.5$.

## 5 Synthesis of cost-minimal adapters

For the cost model $\mathcal{T}$, we solved the problem to synthesize a cost-minimal partner in [9]. We use the same mechanism to synthesize a cost-minimal adapter: Two services $N_1$, $N_2$, an engine $E$, and a cost fumction $\mathcal{F}$ serve as input. The engine $E$ may be computed from an SEA by the tool Marlene[1]. We first compose $N_1$, $N_2$, and $E$ to the service $N = N_1 \oplus E \oplus N_2$. Then, we synthesize a cost-minimal partner for $N$ w.r.t. $\mathcal{F}$.

We sketch the synthesis approach. A central concept of the synthesis is the *minimal budget* of a service $N$ w.r.t. a cost function $\mathcal{F}$. That is, the unique costs of the cost-minimal partner. For $\mathcal{T}$, the minimal budget is either a natural number, or infinite.

In a first step, we find an upper bound $b$ for the minimal budget with the following property: $b$ is infinite iff the minimal budget is infinite. The upper

[1] Marlene is available at http://service-technology.org/marlene [Last accessed on 2012-14-02]

bound $b$ is found by examining $N$ together with its most-permissive partner. Intuitively, the most-permissive partner simulates each other partner of $N$. The most-permissive partner is computed with the tool Wendy [3]. If $b$ is infinite, every partner of $N$ is cost-minimal.

If $b$ is finite, we find the cost-minimal partner by iteration: For a given budget $k$, it is possible to check whether there exists a partner with costs $k$. If such a partner exists, it can be synthesized. This is realized by first transforming $N$ to a service $N^k_{\mathcal{F}}$, and then synthesizing a partner for $N^k_{\mathcal{F}}$. We find the lowest $k \leq b$, such that there exists a partner with costs $k$. This is implemented as a binary search. The resulting partner is by construction cost-minimal.

We implemented this procedure prototypically in Tara[2]. For other cost models, we do not have a solution yet.

## 6 Related work

Seguel et al. [8] study the problem to create *minimal adapters* to resolve deadlock between services. Thereby, minimality is defined w.r.t. to the number of messages considered. The resulting adapter resolves deadlocks and is minimal w.r.t. messages. We study the more general criterion of weak termination. We showed that the minimal adapter is not necessarily cost-minimal. We reduced the synthesis of a cost-minimal adapter to the synthesis of a cost-minimal partner. Zeng et al. [13] use integer programming to find an optimal composite service. That is, an optimal composition of atomic tasks each implemented by a web service. The services do not communicate based on its state, whereas we consider stateful services. De Paoli et al. [4] propose a similar approach for WS-BPEL [1] processes. Both approaches work on well-structured services, whereas we support arbitrary services. The *CLAM framework*, introduced by Zengin et al. [14], combines several adaptation approaches of different layers in one tool to cope with different concerns. It remains open how our approach fits into such a framework. In [9], we presented our results utilizing the partner synthesis by Wolf [11] as a basis. Instead, one could build our approach on top on other synthesis approaches, for instance [7].

## 7 Conclusion

In this paper, we described the problem of cost-minimal adaptation. We suggested a framework to express preference between different adapters based on cost models and cost functions. We discussed two cost models: Worst-case total costs, and worst-case average costs. We sketched the synthesis procedure for worst-case total costs.

We stucture our intended future work as follows: (1) Identification and classification of further cost models, (2) solving partner and thus adapter synthesis for other cost models than $\mathcal{T}$, (3) evaluating the complete approach. For (1) we

[2] Tara is available at http://service-technology.org/tara [Last accessed on 2012-14-02]

plan to further investigate the literature on formalisms which cope with costs in general, for instance, weighted automata. Additionally, we plan to consider results from decision theory. To tackle (2), we will start to develop an algorithm which decides whether one adapter is to be prefered over an other. Then, we extend this result to synthesis of a partner. Part (3) could be realized by a case study with our prototype on real world services.

## References

1. Alves, A., et al.: Web Services Business Process Execution Language Version 2.0. OASIS Standard, 11 April 2007, OASIS (Apr 2007)
2. Gierds, C., Mooij, A.J., Wolf, K.: Reducing adapter synthesis to controller synthesis. IEEE Transactions on Services Computing 99(PrePrints) (2010)
3. Lohmann, N., Weinberg, D.: Wendy: A tool to synthesize partners for services. In: PETRI NETS 2010. pp. 297–307. LNCS 6128, Springer (2010), tool available at http://service-technology.org/wendy [Last accessed on 2012-14-02].
4. Paoli, F.D., Lulli, G., Maurino, A.: Design of quality-based composite web services. In: ICSOC. pp. 153–164 (2006)
5. Papazoglou, M.: What's in a service? In: Oquendo, F. (ed.) Software Architecture, Lecture Notes in Computer Science, vol. 4758, pp. 11–28. Springer Berlin / Heidelberg (2007)
6. Papazoglou, M.P.: Web Services: Principles and Technology. Pearson - Prentice Hall, Essex (Jul 2007)
7. Pistore, M., Traverso, P., Bertoli, P., Marconi, A.: Automated synthesis of composite bpel4ws web services. In: Proceedings of the IEEE International Conference on Web Services. pp. 293–301. ICWS '05, IEEE Computer Society, Washington, DC, USA (2005)
8. Seguel, R., Eshuis, R., Grefen, P.: Constructing minimal protocol adaptors for service composition. In: Proceedings of the 4th Workshop on Emerging Web Services Technology. pp. 29–38. WEWST '09, ACM, New York, NY, USA (2009)
9. Sürmeli, J.: Synthesizing cost-minimal partners for services. Informatik-Berichte 239, Humboldt-Universität zu Berlin (2012), http://u.hu-berlin.de/suermeli-techreport, [Last accessed on 2012-14-02], in publication queue
10. Van Der Aalst, W.M.P.: The application of petri nets to workflow management. The Journal of Circuits Systems and Computers 8(1), 21–66 (1998)
11. Wolf, K.: Does my service have partners? LNCS ToPNoC 5460(II), 152–171 (Mar 2009), special Issue on Concurrency in Process-Aware Information Systems
12. Yellin, D.M., Strom, R.E.: Protocol specifications and component adaptors. ACM Trans. Program. Lang. Syst. 19, 292–333 (March 1997)
13. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for web services composition. IEEE Trans. Software Eng. 30(5), 311–327 (2004)
14. Zengin, A., Marconi, A., Pistore, M.: Clam: cross-layer adaptation manager for service-based applications. In: Proceedings of the International Workshop on Quality Assurance for Service-Based Applications. pp. 21–27. QASBA '11, ACM, New York, NY, USA (2011)