Analyzing Compatibility of Services via Resource Conformance\

Irina A. Lomazova, Ivan V. Romanov

National Research University Higher School of Economics,
Moscow, 101000, Russia
ilomazova@hse.ru; romanov.ekb@gmail.com

Abstract. In this work we consider modeling of services with workflow modules, which form a Petri net subclass. The service compatibility problem is to answer the question, whether two services fit together, i.e. whether the composed system is correct. We study complementarity of resources, produced/consumed by two services — a necessary condition for the service compatibility. Resources, which are produced/consumed by a service, are represented as a multiset language. We define an algebra of multiset languages and present algorithms for checking conformance of resources for two given well-structured workflow modules.

1. Introduction

Service-Oriented Computing is an emerging computing paradigm that supports the modular design of software systems. Complex systems are designed by composing less complex systems, called services.

A service consists of a control structure describing its behavior and of an interface to communicate asynchronously with other services. An interface is a set of (input and output) channels. In order that two services can interact with each other, an input channel of one service should be connected with an output channel of the other service.

The problem of checking service compatibility draws attention of many researchers. A lot of different approaches are being developed to verify correctness of service composition, among them are finite state machine [5, 7], process algebras [16], and Petri Nets [3, 9, 12, 14, 15, 17, 18, 19]. Some researches deal with concrete Web service specifications, such as business process execution language for Web services (BPEL) [19].
We use Petri nets as an underlying formalism. Petri nets offer a wide range of modeling primitives: sequential and parallel composition, choice, massage exchange, expressed in a clear graphical notation, as well as rich analysis methods and tools.

Checking semantical correctness, e.g. deadlock freedom, for composition of two services is a hard problem. Even when a property is decidable, its complexity makes it almost intractable. So, finding relatively easy to check necessary conditions for correctness of services composition may help to find some bugs for low costs and avoid further verification. In this paper we study such a necessary condition — resource conformance of services, notably whether two services have complementary runs, such that all outputs of one service are consumed by and enough for another service and vice versa. The notion of resource conformance is rather week, it does not take into account branching, and even events order. We just ”count resources”, and this necessary condition can be checked in polynomial (on the size of models) time.

Following [15] services are modeled with workflow modules, also called open workflow nets — WF-nets (see e.g. [1]) with additional input/output places representing input/output channels (cf. [3, 4, 8, 9]). The core WF-net in a workflow module describes service control flow, and resource places represent its interface.

Since we study services compatibility we suppose control workflow nets to be sound WF-nets. The soundness property guarantees proper termination of autonomous workflow processes (not taking modules interactions into account, so that resources can be generated or consumed during a process execution without any restrictions). Moreover, we restrict the model to structured WF-modules — an important subclass of workflow modules with control WF-nets sound by construction.

The paper is organized as follows. Section 2 contains some basic definitions and notions, including formal definitions of workflow nets, workflow modules and composition of workflow modules. In Section 3 a motivating example of two workflow modules, modeling a part of credit allowence system, is given. In Section 4 we define and study a language of quasi regular expressions for describing a workflow module resource interfaces. In Section 5 we present an algorithm for checking resource compatibility of two structured workflow modules. Section 6 contains some conclusions.

2. Petri nets and workflow modules. Definitions

Let $S$ be a finite set. A multiset $m$ over a set $S$ is a mapping $m : S \rightarrow \text{Nat}$, where $\text{Nat}$ is the set of natural numbers (including zero), i. e. a multiset may contain several copies of the same element.

For two multisets $m, m'$ we write $m \subseteq m'$ iff $\forall s \in S : m(s) \leq m'(s)$ (the inclusion relation). The sum and the union of two multisets $m$ and $m'$ are defined as usual: $\forall s \in S : m + m'(s) = m(s) + m'(s)$, $m \cup m'(s) = \max(m(s), m'(s))$. By $M(S)$ we denote the set of all finite multisets over $S$.

Non-negative integer vectors are often used to encode multisets. Actually, the set of all multisets over finite $S$ is a homomorphic image of $\text{Nat}^{|S|}$.

Let $P$ and $T$ be disjoint sets of places and transitions and let $F : (P \times T) \cup (T \times P) \rightarrow \text{Nat}$. Then $N = (P, T, F)$ is a Petri net. A marking in a Petri net is a function $M : P \rightarrow \text{Nat}$, mapping each place to some natural number (possibly zero). Thus a marking may be considered as a multiset over the set of places. Pictorially, $P$-elements are represented by circles, $T$-elements by boxes, and the flow relation $F$ by directed arcs. Places may carry tokens represented by filled circles. A current marking
$M$ is designated by putting $M(p)$ tokens into each place $p \in P$. Tokens residing in a place are often interpreted as resources of some type consumed or produced by a transition firing.

For a transition $t \in T$ an arc $(x,t)$ is called an input arc, and an arc $(t,x)$ — an output arc; the preset $\cdot t$ and the postset $t^\bullet$ are defined as the multisets over $P$ such that $\cdot t(p) = F(p,t)$ and $t^\bullet(p) = F(t,p)$ for each $p \in P$.

A transition $t \in T$ is enabled in a marking $M$ iff $\forall p \in P \ M(p) \geq F(p,t)$. An enabled transition $t$ may fire yielding a new marking $M' = M - \cdot t + t^\bullet$, i.e. $M'(p) = M(p) - F(p,t) + F(t,p)$ for each $p \in P$ (denoted $M \xrightarrow{t} M'$, or just $M \rightarrow M'$). We say that $M'$ is reachable from $M$ iff there is a sequence of firings $M = M_1 \rightarrow M_2 \rightarrow \cdots \rightarrow M_n = M'$. For a Petri net $N$ by $R(N,m)$ we denote the set of all markings reachable in $M$ from the marking $m$, by $R(N,m)$ — the set of all markings reachable in $M$ from its initial marking.

Workflow nets (WF-nets) [1] is a special subclass of Petri nets designed for modeling workflow processes. A workflow net has one initial and one final place, and every place or transition in it is on a directed path from the initial to the final place.

**Definition 2.1.** A Petri net $N$ is called a workflow net (WF-net) iff

1. There is one source place $i \in P$ and one sink place $f \in P$ s.t. $\cdot i = f^\bullet = \emptyset$;

2. Every node from $P \cup T$ is on a path from $i$ to $f$.

3. The initial marking in $N$ contains the only token in its source place.

By abuse of notation we denote by $i$ both the source place and the initial marking in a WF-net. Similarly, we use $f$ to denote the final marking in a WF-net $N$, defined as a marking containing the only token in the sink place $f$.

An important correctness property for WF-nets is soundness. Classical WF-nets are called sound if one can reach the final marking from any marking reachable from the initial marking. The intuition behind this notion is that no matter what happens, there is always a way to complete the execution and reach the final state. This soundness property is sometimes also called proper termination and corresponds to classical soundness in [2].

**Definition 2.2.** A WF net $N$ with a source place $i$ and a sink place $f$ is called sound iff

1. For every marking $m$ reachable from the initial marking $i$, there exists a firing sequence leading from $m$ to the final marking $f$:

   $\forall m \in R(N) : (i \xrightarrow{*} m) \Rightarrow (m \xrightarrow{*} f)$;

2. The marking $f$ is the only marking reachable from $i$ with at least one token in place $f$:

   $\forall m \in R(N) : m \geq f \Rightarrow m = f$;

3. There are no dead transitions in $N$:

   $\forall t \in T \exists m, m' : i \xrightarrow{t} m \rightarrow m'$. 
Figure 1. Routing operations: (a) sequential routing, (b) parallel routing, (c) iteration, (d) conditional routing

For an arbitrary WF net soundness is decidable, but it is EXPSPACE-hard [6].

Modeling workflow consists of modeling case management with the help of sequential routing, parallelism, iteration, and conditional routing. To express it explicitly building blocks such as the AND-split, AND-join, OR-split and OR-join can be used. The AND-split and the AND-join are used for parallel routing. The OR-split and the OR-join are used for conditional routing. All these constructs can be easily expressed in Petri net formalism.

To guarantee, that we get 'good' workflows, we are to balance AND/OR-splits and AND/OR-joins. Clearly, two parallel flows initiated by an AND-split, should not be joined by an OR-join. Two alternative flows created via an OR-split, should not be synchronized by an AND-join. When we follow these rules we obtain well-structured WF-nets (see [1] for more details).

Fig. 1 shows fore routing operations used in structured WF-nets. Thus, to apply sequential operation to two WF-nets $N_1$ and $N_2$ is to substitute $N_1$ for $t_1$, and $N_2$ for $t_2$ in the net, shown in Fig. 1 (a), by substituting the source place of $N_1$ for $p_1$, merge the sink place of $N_1$ and the source place of $N_2$ (for $p_2$), and substitute the sink place of $N_2$ for $p_3$. To apply parallel operation to WF-nets $N_1$ and $N_2$ is to substitute $N_1$ for $t_2$, and $N_2$ for $t_3$ in the net, shown in Fig. 1 (b), while transitions $t_1, t_4$ are additional routing transitions here. The iteration and conditional operations are defined in the similar way.

**Definition 2.3.** An atomic WF-net is a WF-net, consisting of one source place $i$, one sink place $f$ and
one transition, for which \( i \) is the only input place, and \( f \) is the only output place.

A WF-net is called a \textit{well-structured WF-net} iff it can be obtained from atomic WF-nets by successive application of routing operations, shown in Fig. 1.

It was shown in [1], that well-structured WF-nets are sound by construction.

To model services we use workflow modules — a special subclass of Petri nets.

\textbf{Definition 2.4.} A Petri net \( M = (P, T, F) \) is called a workflow module (WF-module) iff

1. The set \( P \) of places is a disjoint union of three sets: internal places \( P^N \), input places \( P^I \), and output places \( P^O \).

2. The flow relation is divided into internal flow \( F^N \subseteq (P^N \times T) \cup (T \times P^N) \) and communication flow \( F^C \subseteq (P^I \times T) \cup (T \times P^O) \).

3. The net \( PM = (P^N, T, F^N) \) is a WF-net.

4. No transition is connected both to an input place and an output place.

Within a WF-module \( M \), the workflow net \( PM \) is called the \textit{internal process} of \( M \) and the tuple \( \mathcal{I}(M) = (P^I, P^O) \) is called its \textit{interface}. Places belonging to the interface \( \mathcal{I}(M) \) are called \textit{ports}. We suppose that each port in \( \mathcal{I}(M) \) has a unique name. A workflow module is called \textit{well-structured} iff its internal process is a well-structured WF-net.

Fig. 2 shows an example of a WF-module \( WFM_1 \), describing a simple model of a bank service. Here \( p_1 \) is a source place, \( p_8 \) — a sink place. Places \( p_1, \ldots, p_8 \) are internal places. The interface of \( M_1 \) consists of input places \( CR \) and \( A \), and output places \( CD \) and \( E \). The communication flow of \( WFM_1 \) includes arcs \((t_1, CD)\), \((CR, t_3)\), \((A, t_7)\), and \((t_9, E)\). All other arcs belong to the internal process of \( M_1 \). Note, that \( WFM_1 \) is a well-structured WF-module.

\textbf{Definition 2.5.} A WF-module is called \textit{sound} iff its internal process is sound.

A composition of WF-modules, modeling Web services interaction, is defined as follows.
Definition 2.6. Let $M_1, M_2$ be two WF-modules. A composition $M_1 \odot M_2$ of $M_1$ and $M_2$ is a net $N$ obtained from $M_1$ and $M_2$ by merging input ports of $M_1$ with output ports of $M_2$ with the same names, and similarly output ports of $M_1$ with input ports of $M_2$, i.e. $p_1 \in \mathcal{I}(M_1)$ and $p_2 \in \mathcal{I}(M_2)$ are merged iff they have the same name and one of them is an input port, while the other is an output port.

An example of composing two WF-modules is shown in the next section.

A composition $M_1 \odot M_2$ of two WF-modules $M_1$ and $M_2$ can be easily transformed into a WF-module as follows. Let $i_1, i_2$ be source places in $M_1$ and $M_2$ correspondingly, and $f_1, f_2$ be their output places. Add to $M_1 \odot M_2$ a new source place $i$, a new sink place $f$, and two new transitions $t_i, t_f$, s.t. $t_i = \{i\}$, $t_f = \{i_1, i_2\}$, and $t_\ast_f = \{f\}$, $\ast t_i = \{f_1, f_2\}$.

We say that a composition $M_1 \odot M_2$ is sound iff its corresponding WF-module is sound.

3. Motivating example

As an illustrating example we consider a model of credit allowance. The model describes two services, represented by two WF-modules.

The WF-module $WFM_1$ in Fig. 2 models a service of a bank credit system, which estimates whether it is reasonable to loan money to this or that person. According to this model the procedure starts from the initial state, when the only token resides in the place $p_{11}$. The service sends a client data (CD) to the credit bureau ($t_1$) and 'stays' in $p_{12}$ waiting for a response. After receiving ($t_3$) the credit rating (CR) and analyzing it the service either fires $t_2$ to repeat the request, or choses one of two possible variants of payment. The first variant (via $t_4$) demands paying accounts (A) and then sending a report (E). The second variant specifies paying a loan ($t_5$), then sending new client data ($t_8$), and after that sending a report ($t_{10}$).

Fig. 3 shows a WF-module, modeling a credit bureau. It is a company that collects information from various sources and provides consumer credit information on individual consumers for a variety of users. The model includes four internal places and four interface places. The place $p_{10}$ is the source place, representing the initial state, when the service is waiting for the client information from the external bank system. When the service receives data from CD, the transition $t_{11}$ fires, meaning a client credit rating is calculated, and a token goes to $p_{12}$. If a bank requests for a credit rating (CR), then $t_{12}$ fires meaning that the credit bureau sends requested information. After that the service sends an account (A) to the bank and returns to the initial state. Otherwise, if the credit bureau receives the end request (E), it goes to the final state ($p_{13}$) and stops.
Interaction of these two services is modeled by the composition of the corresponding WF-modules, depicted in Fig. 4. Services communicate with each other in the following way: the bank sends all information about the person to the credit bureau, this information is analyzed, and basing on all data credit bureau provides a credit rating of a definite client to the bank. This sequence of actions may be repeated several times.

While each of WF-modules $WFM_1$ and $WFM_2$ is sound, their composition $WFM_1 \odot WFM_2$ is not sound. The reason of this is that the services are resource incompatible, i.e. for these services a joint execution without pending inputs and/or outputs is not possible. In the next sections we show, how to detect such incompatibilities.

4. **Workflow module resources and multiset languages**

A multiset over a given set of objects corresponds to a string over an alphabet of object names (disregarding ordering of letters in words). A multiset language (a macroset) \cite{10, 11} is a finite, or infinite set of multisets over some finite alphabet. In his works M. Kudlek studied families of multiset languages, generated by multiset grammars. He extended to multiset languages usual language operations, and investigated closure properties of main classes of multiset languages.

Here we in some way continue ‘translation’ of notions, related to sequential formal languages, to multiset languages. We define quasi-regular expressions, generating a class of multiset languages, and use them for describing resources, produced/consumed by well-structured workflow nets. Identification of workflow resources is employed for checking resource conformance, or similarity of workflow modules.

For describing resources of well-structured workflow modules we define a special language of Quasi-Regular Expressions (QRE). Quasi-regular expressions are constructed from atoms extended by a special...
symbol $\epsilon$ for the empty multiset by applying three operations: $\oplus$ corresponds to multiset union, $\circ$ is a kind of multiset concatenation, and $^*$ is an analog of Kleene star (iteration).

**Definition 4.1.** Let $\text{Atom}$ be a finite set of atoms (letters). The language $QRE = QRE(\text{Atom})$ of quasi-regular expressions over $\text{Atom}$ is defined by induction as follows:

1. $\epsilon \in QRE$;
2. $a \in QRE$, if $a \in \text{Atom}$;
3. $e_1 \circ e_2 \in QRE$, if $e_1, e_2 \in QRE$;
4. $e_1 \oplus e_2 \in QRE$, if $e_1, e_2 \in QRE$;
5. $e^* \in QRE$, if $e \in QRE$.

Semantics of $QRE$ maps each quasi-regular expression $e \in QRE(\text{Atom})$ to a multiset language $\mu(e)$, i.e. a set of multisets over $\text{Atom}$ according to the following rules:

1. $\mu(\epsilon) = \emptyset$;
2. $\mu(a) = [a]$ for $a \in \text{Atom}$;
3. $\mu(e_1 \circ e_2) = \{m_1 + m_2 | m_1 \in \mu(e_1), m_2 \in \mu(e_2)\}$;
4. $\mu(e_1 \oplus e_2) = \mu(e_1) \cup \mu(e_2)$;
5. $\mu(e^*) = \mu(e^0) \cup \mu(e^1) \cup \mu(e^2) \cdots \cup \mu(e^n) \cdots$, where $e^0 = \epsilon$, $e^n = e \circ e^{n-1}$ for $n \geq 1$.

Now we say, that two quasi-regular expressions $e_1$ and $e_2$ are equivalent (denoted $e_1 = e_2$ by abuse of notation) iff $\mu(e_1) = \mu(e_2)$. The next theorem defines the algebra of quasi-regular expressions.

**Theorem 4.1.** For all quasi-regular expressions $e, e_1, e_2, e_3 \in QRE$ the following equations are valid:

1. $e \circ e = e$,
2. $e \oplus e = e$,
3. $e^* = e$,
4. $e_1 \circ e_2 = e_2 \circ e_1$,
5. $e_1 \oplus e_2 = e_2 \oplus e_1$,
6. $(e_1 \circ e_2) \circ e_3 = e_1 \circ (e_2 \circ e_3)$,
7. $(e_1 \oplus e_2) \oplus e_3 = e_1 \oplus (e_2 \oplus e_3)$,
8. $e \circ (e_1 \oplus e_2) = (e \circ e_1) \oplus (e \circ e_2)$,
9. $e \oplus e = e$, 

10. \((e^*)^* = e^*\),

11. \((e_1 \oplus e_2)^* = e_1^* \circ e_2^*\),

12. \((e_1 \circ e_2)^* = e_1^* \circ e_2^*\).

**Proof:**
Checking these equivalences is quite straightforward. By way of example we prove equation 11 from the list, i.e. \(\mu((e_1 \oplus e_2)^*) = \mu(e_1^* \circ e_2^*)\).

First we show, that \(\mu((e_1 \oplus e_2)^*) \subseteq \mu(e_1^* \circ e_2^*)\). Let \(m \in \mu((e_1 \oplus e_2)^*)\). Then for some \(n \geq 0\) \(m \in \mu(e_1 \oplus e_2)^n\), i.e. \(m = m_1 + \cdots + m_n\), where \(m_1, \ldots, m_n \in \mu(e_1 \oplus e_2) = \mu(e_1) \cup \mu(e_2)\). By commutativity and associativity of multiset union we can think \(m = (m_1 + \cdots + m_k) + (m_{k+1} + \cdots + m_n)\), where \(m_1, \ldots, m_k \in \mu(e_1)\), and \(m_{k+1} + \cdots + m_n \in \mu(e_2)\) \((0 \leq k \leq n)\). Since \(m_1 + \cdots + m_k \in \mu(e_1)\), and \(m_{k+1} + \cdots + m_n \in \mu(e_2)\), we get \(m = (m_1 + \cdots + m_k) + (m_{k+1} + \cdots + m_n) \in \mu(e_1^* \circ e_2^*)\).

Now we show, that \(\mu(e_1^* \circ e_2^*) \subseteq \mu((e_1 \oplus e_2)^*)\). Let \(m \in \mu(e_1^* \circ e_2^*)\). Then there exist \(n_1, n_2 \geq 0\) such that \(m = m_1 + m_2\), where \(m_1 \in \mu((e_1)^{n_1})\), and \(m_2 \in \mu((e_2)^{n_2})\). That means \(m_1 = m_{11} + \cdots + m_{1n_1}\), and \(m_2 = m_{2} + \cdots + m_{2n_2}\), where \(m_{11}, \ldots, m_{1n_1} \in \mu(e_1)\), and \(m_{2}, \ldots, m_{2n_2} \in \mu(e_2)\). Hence \(m = m_{11} + \cdots + m_{1n_1} + m_{2} + \cdots + m_{2n_2}\), where \(m_{11}, \ldots, m_{1n_1}, m_{2}, \ldots, m_{2n_2} \in \mu(e_1) \cup \mu(e_2) = \mu(e_1 \oplus e_2)\), and thus \(m \in \mu((e_1 \oplus e_2)^{n_1+n_2})\). This implies \(m \in \mu((e_1 \oplus e_2)^*)\). \(\square\)

**Definition 4.2.** We say that a quasi-regular expression \(e \in QRE\) is in a standard form, iff
\[ e = e_1 \oplus \cdots \oplus e_n, \]
where \(e_1, \ldots, e_n\) do not contain multiset union \(\oplus\) and nested iteration \(^*\) \((n \geq 1)\).

**Theorem 4.2.** Every quasi-regular expression can be transformed to an equivalent quasi-regular expression in the standard form by applying equations (1–12).

**Proof:**
To take \(\oplus\)-operation outside we use equations 8 and 11. Equations 10, 12 allow taking \(\circ\)-operation outside in subexpressions without \(\oplus\). Nested \(^*\) are eliminated by using equation 10. \(\square\)

Let now \(M\) be a well-structured WF-module with an interface, consisting of two disjoint sets \(P^I\) (input places) and \(P^O\) (output places). Consider then some run \(\delta\) for \(M\) — a sequence of states and transition firings, starting from the initial state and coming to the final state of \(M\). For each run \(\delta\) the pair of input and output resources \(R(\delta) = (R^I(\delta), R^O(\delta))\), where \(R^I(\delta) \in M(P^I)\) and \(R^O(\delta) \in M(P^O)\), are defined as multisets of inputs/outputs consumed/produced in the course of \(\delta\) execution. By \(\rho(M)\) we denote the set of all pairs of resources for all runs in \(M\). More formally:

**Definition 4.3.** Let \(M\) be a structured WF-module with input ports \(P^I\) and output ports \(P^O\). Then for \(R^I \in M(P^I)\) and \(R^O \in M(P^O)\) we define \((R^I, R^O) \in \rho(M)\) iff \(f + R^O \in R(M, i + R^I)\).
It is easy to note, that a pair of multisets over non-intersecting sets of places can be considered as just one multiset. Thus for a workflow module a resource is a multiset language.

Now we show, that for a well-structured WF-module $M$ a quasi-regular expression $e(M)$, describing interface resources $\rho(M)$, can be effectively constructed. To make expressions more readable we prefix names of input resources by the query mark ‘?’ and output resources — by exclamation mark ‘!’.

Recall, that well-structured workflow nets are constructed from atomic transitions by sequential application of four control structures: sequential routing, conditional routing, parallel routing and iteration (see Fig. 1).

Algorithm 4.1. (Constructing quasi-regular expressions representing interface resources of well-structured WF-modules).

For a structured workflow module $M$ a quasi-regular expression $e(M)$ can be constructed by induction on the structure of internal process $N$ of $M$:

- for an atomic net $N$ — a transition with input resource places $p_1, \ldots, p_k$ and output resource places $q_1, \ldots, q_n$, define $e(N) = ?p_1 \circ \cdots \circ ?p_k \circ !q_1 \circ \cdots \circ !q_n$;
- for $N$ being sequential composition of $N_1$ and $N_2$ define $e(N) = e(N_1) \circ e(N_2)$;
- for $N$ being parallel composition of $N_1$ and $N_2$ define $e(N) = e(N_1) \circ e(N_2)$;
- for $N$ being selective composition of $N_1$ and $N_2$ define $e(N) = e(N_1) \oplus e(N_2)$;
- for $N$ being an iteration of $N_1$ and $N_2$ define $e(N) = e(N_1) \circ (e(N_2) \circ e(N_1))^*$.

Theorem 4.3. Let $M$ be a structured WF-module, and let $e(M)$ be a quasi-regular expression, obtained for $M$ according to the Algorithm 1. Then $e(M) = \rho(M)$.

The proof of this theorem is straightforward by induction on the structure of the internal process of a given WF-module.

5. Resource similarity and compatibility

We say that two workflow modules are resource similar, if they produce/consume the same resources in the course of their runs. We say that two workflow modules are resource compatible, if they may execute runs with producing/consuming mutually complementary resources. If the composition of two workflow modules is sound, then they are resource compatible. However, it could be that for resource compatible modules their composition is not sound, since resources may be outputted in a wrong order. So, resource compatibility is a necessary, but not sufficient condition for the correctness of services composition.

Definition 5.1. Let $M_1, M_2$ be two WF-modules with source places $i_1, i_2$ and sink places $f_1, f_2$ correspondingly.

- $M_1$ and $M_2$ are resource similar iff $\rho(M_1) = \rho(M_2)$;
- $M_1$ and $M_2$ are resource compatible iff $f_1 + f_2 \in R(i_1 + i_2)$ in $M_1 \circ M_2$. 
Immediately from the definition we get that if $M_1 \odot M_2$ is sound, then $M_1$ and $M_2$ are resource compatible. Note, that resource similarity is a rather weak equivalence. It could be that modules $M_1$ and $M_2$ are resource compatible, $M'_1$ is resource similar to $M_1$, but $M'_1$ and $M_2$ are not resource compatible. However, if $M_1$ and $M_2$, as well as $M'_1$ and $M_2$ are resource compatible, then $M_1$ and $M'_1$ are resource similar.

Both resource similarity and resource compatibility are decidable, since these problems can be reduced to similar problems for classical regular expressions. More important is that resource compatibility can be checked in polynomial time.

We present now an effective algorithm for checking resource compatibility for well-structured WF-modules.

**Algorithm 5.1. (checking resource compatibility of two well-structured WF-modules).**

Let $M_1, M_2$ be well-structured WF-modules.

**Step 1.** Construct quasi-regular expressions $e(M_1), e(M_2)$.

**Step 2.** Reduce $e(M_1), e(M_2)$ to standard forms $e_s(M_1), e_s(M_2)$ correspondingly.

**Step 3.** Construct a complementary expression $e_s^{-1}(M_1)$ by changing $?$ for $!$ and vice versa in $e_s(M_1)$.

**Step 4.** Check whether $\mu(e_s^{-1}(M_1)) \cap \mu(e_s(M_2))$ is empty. If $\mu(e_s^{-1}(M_1)) \cap \mu(e_s(M_2))$ is empty, output “NO”, otherwise output “YES”.

All steps of the Algorithm except for Step 4 are already described. Step 4 is to check, given two quasi-regular expressions $e$ and $e'$ in regular form, whether $\mu(e) \cap \mu(e')$ is empty. This can be reduced to solving a set of linear equations.

We illustrate this algorithm by applying it to our motivation example from Section 3. First we build quasi-regular expressions for bank and credit bureau services:

$e(WFM_1) = (\lnot CD \circ ?CR) \circ (\lnot A^{\circ} E) \oplus (\lnot CD \circ ?E);$  
$e(WFM_2) = \lnot CD \circ (\lnot CR \circ ?CD)^{*} \circ ?E.$

Then we are to reduce these expressions to regular forms. Note, that $e(WFM_2)$ is already in a regular form, and hence $e(WFM_2) = e_s(WFM_2)$. To reduce $e(WFM_1)$ it is sufficient to apply distributivity law (equation 8):

$e_s(WFM_1) = ((\lnot CD \circ ?CR)^{*} \circ ?A^{\circ} E) \oplus ((\lnot CD \circ ?CR)^{*} \circ CD \circ ?E).$

Then

$e_s^{-1}(WFM_1) = ((\lnot CD \circ ?CR)^{*} \circ A^{\circ} ?E) \oplus ((\lnot CD \circ ?CR)^{*} \circ ?CD \circ ?E).$

Now we have

$r \in e_s^{-1}(WFM_1)$ iff $r = n_1 \cdot \lnot CD + n_1 \cdot !CR + n_2 \cdot !A + ?E,$

or

$r = (n_1 + 1) \cdot \lnot CD + n_1 \cdot !CR + ?E$ for some natural $n_1, n_2 \geq 0.$

Similarly,

$r \in e_s(WFM_2)$ iff $r = (n + 1) \cdot \lnot CD + n \cdot !CR + n \cdot !A + ?E$ for some natural $n \geq 0.$
The multiset \( \mu(e^{-1}_s(WFM_1)) \cap \mu(e_s(WFM_2)) \) is not empty iff at least one of the following sets of simultaneous equations is consistent.

**Set 1 of equations** (corresponds to the first summand in \( e_{s-1}(WFM_1) \)):
- \( n_1 = n + 1 \) (factors of \(?CD?)
- \( n_1 = n \) (factors of \!CR)
- \( n_2 = n \) (factors of \(!A?)
- \( 1 = 1 \) (factors of \?E)

**Set 2 of equations** (corresponds to the second summand in \( e_{s-1}(WFM_1) \)):
- \( n_1 = n + 1 \) (factors of \(?CD?)
- \( n_1 = n \) (factors of \!CR)
- \( 0 = n \) (factors of \(!A?)
- \( 1 = 1 \) (factors of \?E)

Both these sets of equations are obviously inconsistent. So, we can immediately conclude, that the services of our bank example in Section 3 are not compatible.

**Theorem 5.1.** Let \( M_1, M_2 \) be two structured WF-modules. Modules \( M_1 \) and \( M_2 \) are resource compatible iff the output of the Algorithm 5.1 for \( M_1, M_2 \) is “YES”.

The proof of this theorem is rather straightforward. It is based on the previous results and the definition of QRE semantics.

Note, that Algorithm 2 has a polynomial on the size of WF-module time complexity. Constructing quasi-regular expression for a given WF-module can be done by finding hammocks in its internal WF-net, since hammocks correspond to well-structured subnets in a well-structured WF-nets (cf. [13]). Finding all hammocks in a given WF-net with \( k \) nodes can be solved in time \( O(k^4) \). Reducing a quasi-regular expression to a standard form will also take not more then \( O(k^4) \) steps. Constructing a complementary expression can be done in a linear time, and solving linear equations by e.g. Gaussian elimination takes also not more then \( O(k^4) \) steps. Thus resource compatibility of two WF-modules with \( k \) nodes can be checked in time \( O(k^4) \).

**6. Conclusion**

In this paper we have presented a new approach for detecting incompatibility of services by analysis of resource conformance of their structured workflow models. The resource compatibility is a necessary condition for soundness of composition of workflow models.

We have introduced a language of quasi-regular expressions for describing multiset languages, and use this language for representing interface resources of structured workflow modules. Checking resource compatibility is then reduced to checking emptiness of intersection of multiset languages, represented by quasi-regular expressions. Thus checking resource compatibility can be solved in polynomial time.
Though resource compatibility is not sufficient for correct service interaction, the proposed procedure may be helpful for efficient detecting inconsistencies on the preliminary stages of service analysis. Note that known methods for checking service compatibility need exponential time (cf. [12]).

Presented here results can be easily extended to checking compatibility of more than two WF-modules, when resource compatibility means that all resources produced within one run by a set of modules are consumed by these modules within the same run.

References


