Accepted Manuscript

The Reviews and Analysis of the State-of-the-Art Service Workflow Specification Languages

Wattana Viriyasitavat, Andrew Martin

PII: S2452-414X(17)30035-3
Reference: JII 44

To appear in: Journal of Industrial Information Integration

Received date: 27 June 2017
Revised date: 19 July 2017
Accepted date: 24 July 2017


This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.
The Reviews and Analysis of the State-of-the-Art Service Workflow Specification Languages

Wattana Viriyasitavat¹, Andrew Martin²

¹Business Information Technology Division, Department of Statistics, Chulalongkorn University, Pathumwan, Bangkok, 10330, Thailand,
²Department of Computer Science, University of Oxford, Oxford, OX1 3 PR United Kingdom

Abstract

In the globalized market and business environment, enterprises strive to cope with rapid market changes and competition. To sustain competitiveness, enterprises need to continually adapt their business processes to accommodate changes promptly. This view includes the integration of external services with their internal services for required capabilities to catch new business opportunities. A number of services with similar service functionalities are available, posing a great challenge in composing optimized business processes in a timely manner. This requires effective methodologies and tools in selecting services and composing them as service workflows for specified goals. Service workflow technology facilitates these by providing methodologies to support business process modeling and reengineering to optimize and automate processes according to workflow requirement specifications. This paper provides reviews and analysis of the state-of-the-art service workflow specification languages, including (1) Self-Adaptive Configuration based on HMS (2) BAM, (3) CTR, (4) SOLOIST, and (5) SWSpec. These languages are evaluated against the fundamental principles for general service workflow specification languages.

Keywords: Service; Workflow; Specification Language; Business Processes; Logic; CTL;

1. Introduction

The Internet and Service Oriented Computing (SOC) technologies have revolutionized the ways of businesses management. For enterprises, the boundaries with business environments become ambiguous; enterprises are able to utilize geographically distributed services to implement solutions for their specific goals (Bi et al 2017, Cochran et al 2016, 2017, Wang et al. 2016). Enabled by SOC technology, the paradigms for business collaboration have been evolved from data-driven to process-driven modes (Lin et al 2009, Zhang et al. 2014, Moser et al 2012, Yen et al 2008, Huemer et al 2008). In this context, service workflow is an important tool to model and formulate the coordination and composition of services (Viriyasitavat et al 2012, 2014a, 2014b). The workflow has been widely used to model large-scale scientific and engineering applications in various domains such as earth science, astronomy, physics, healthcare, telecommunications, military, bioinformatics, administration at universities, smart grid, and digital libraries (Anderson et al 2003, Cardoso et al 2004, Deelman et al 2016, Espinoza et al 2013).

Nowadays, the business processes are facing important challenges as they encounter frequently changes from customers, global competition and technological advances. Service workflows are effective methodologies to deal with these challenges (Viriyasitavat and Martin 2012). They rely on services as basic building blocks to construct dynamic, self-adaptable, cost-effective, and optimized solutions to complex business processes, according to certain requirements. Functionalities from inside or outside an organization are viewed as services, which can be orchestrated by service workflows. They have a number of advantages over conventional methods such as easiness of computer representation, reusability, failure management, adaptability, and suitability of parallel computing for fast solutions (Viriyasitavat 2016). Moreover, service workflows reduce the need to develop new components each time a new business process arises. Once an
organization captures new business possibilities, it can reengineer business processes to improve it or adapt it to changing requirements (Georgakopolulos et al 1995). Due to the great potentials, service workflows have been greatly studied and applied by academic and industry sectors.

The objectives of this paper are to provide the reviews and analysis of the state-of-the-art service workflow specification languages, including (1) Self-Adaptive Configuration based on HMS (2) BAM, (3) CTR, (4) SOLOIST, and (5) SWSpec. These languages are evaluated according to the fundamental principles for general service workflow specification (Viriyasitavat and Martin 2011).

2. General Principles in the Development of Workflow Specification Languages

In the past, workflows would be usable if their processes and definitions can capture several business needs, usually confined within an organization. SOC technology enables the workflow applicability to go beyond confined business processes. As already mentioned, its current use is to facilitate inter-organizational interoperations based on services. A sophisticated service workflow can compose of several delegated sub-services. Nowadays, interoperation in the form of service workflows is prevalent (Viriyasitavat and Martin 2012). Specification languages are key elements for service selection in this context (Viriyasitavat et al 2012).

The fundamental idea of requirement specification language in service workflows is to allow users to formally express requirements in service workflows based on attributes of other services. User stereotypes in this context include workflow owners who specify requirements to control and regulate service selection and composition to form service workflows and service providers to use the language as a means to express requirements for workflow participation. In the past, service selection and workflow structure are manual processes done by workflow owners, with SOC and auxiliary technologies these processes become automatic to respond to dynamic changing environments.

In summary, workflow specification language is a formal tool to express requirements that can be reasoned, verifying attributes of services and validating services to be part of a workflow. Selection decision can be made from the result of manual/automatic compliance checking. Due to the importance of the languages, this section summarized the principles for service workflow specification languages (Viriyasitavat and Martin 2011b), enabling shared understanding among workflow researchers and practitioners.

R1: Interoperability with Local Requirements

In global management, centralized workflow management software is responsible for managing services and workflow structure. Requirements are defined globally and enforced to every service by the owner. In this case, the enforcement is situated at the domain level between the owner and each prospective service. In more complex cases, hierarchical workflows usually deal with service delegation. The delegated services are trusted if they comply with local requirements of their delegators.

Conversely, in a decentralized environment where no single point of control exists, requirements are locally managed, situating from one to other services. Service selection is typically determined by conglomerate of local requirements from each participating service. Workflow specification languages should address these two aspects and be uniformed enabling the same understanding among involved parties.

R2: Separation of Duty (SoD)

Services with complementary competencies are composed to deliver a complex task. SoD guarantees the avoidance of conflict-of-interest situations among composed services. Typically, a service is assigned to a specific role and execution for a particular task, where SoD verifies based on role and execution assignment
(Nyanchama and Osborn, 1999, Jaeger and Tidswell, 2001). Workflow specification languages should be able to address the aspect of SoD requirements.

**R3: The Association between Tasks and Services**

A set of services with complementary capabilities can be composed to perform a complex task. Conversely, one single service is able to execute several tasks in one or several workflows. For example, an online patient record service is responsible for (1) managing patients’ information and (2) providing statistical analysis to the National Health Service. Privacy and confidentiality are two requirements of the first while the later concerns with data accuracy. Consequently, service selection decision is differently determined depending on which tasks they execute. Workflow specification language should be able to specify requirements based on the association of tasks and services.


The structure nature of service workflows contains branches and tasks. Requirements in workflow executions can be categorized in two aspects. The first aspect is flow-oriented. These requirements focus on services involved in a flow execution. For instance, to enable data secrecy, digital patient’s records must be encrypted during transmission among services involved along the flow. The second aspect is task-oriented. Requirements in services selection are specified and validated according to the task they execute. Since one task can be executed by several services, attributes of one service in the task may have an influence on participation decision of other services to join the same task. For instance, the heath-claiming task must be approved by two different services, one from an insurance company and another from a contracted hospital. To complete this task, an insurance and hospital service must be presented. Workflow specification language should accommodate both flow- and task-oriented characteristics.

**R5: Enforcement of Sequences**

In a flow-oriented specification, requirements of a workflow also include the sequence of tasks and services that are dependent on one another. One subsequent task might require the results from others occurred before, as well as to provide results to the services afterwards (Kuntze and Sch, 2008). From the service point of view, it is desirable for workflow specification languages to be able to enforce to the sequence of tasks and service associated. For instance, the task of issuing a check for a tax refund can be done after a clerk and financial manager services have approved in sequence.

**R7: R6: Flexible Degrees of Restriction**

In open and complex environments, one party is usually unable to have full visibility on the whole service workflow elements. The absence of the end-to-end visibility has led workflow research to re-examine and to find the way for workflow cooperation (Falcone et al., 2003). One advantage of flexible degrees of visibility is to enable entities to retain the level of privacy of internal executions. This gives a major concern on how one to accurately specify requirements of other services in a workflow or sub-workflows. Workflow specification languages should be flexible to formally express requirements with several degrees of restriction based on visibility.
R7: Protection of Workflow Data

Since data is traversed along a workflow path, protecting the data against security threats becomes necessary. This requirement has been sufficiently accommodated by protections offered by information security, such as integrity, confidentiality, secrecy, privacy, etc. These requirements are essential to be specified as the required attributes where workflow specification languages should be developed in a way to address them.

3. Workflow Specification Languages


The main motivation of this work is to address dynamic changes of services in a workflow, with the assumption that new services can be discovered dynamically. Holonic Multi-agent Systems (HMS) (Koestler 1990) is employed for dynamic real-time re-configuration for service collaboration. In this paper, the authors study a workflow adaptation problem (WAP) and propose a solution based on contract net protocol (CNP) to solve WAP in the context of service composition based on HMS. CNP is based on Petri Net as a workflow language. The reasons of using Petri Net to model service workflows have been well supported in (Viriyasitavat 2013) and (van der Aalst, 1998). Petri Net-based a workflow language is well-established formal mechanism for modeling, analysis as well as verification of the attributes of composed services. In this work, there are two types of atomic services in a workflow, (1) atomic task and (2) atomic activity.

Atomic Tasks: Specify the operations and precedence constraints of the operations in a task.

Atomic Activities: Capture the capabilities or functions of an actorservice to perform atomic tasks.

Using time Petri Nets to represent the workflow, the process of service replacement when changes occur is relatively straightforward. After new services are discovered, the existing services send notification messages to the involved (connected) services. Then, on receiving the notification, each service will check whether any of its terminal output places is affected by the replaced services. The system will determine the optimal solution to make decision whether the replacement should continue or not. However, the optimization is based on the assumption of minimizing cost, which has no description about what really the cost is. Figure 1 illustrates the association between atomic tasks (circle) and atomic activities (circle with black dot).
Fig. 1. Example of required attributes p, q, and r for task 1 execution
Analysis:

The analysis of this language against the general principles in section 2 is as follows (see Table 1):

R1: Interoperability with Local Requirements
The language does not allow local requirements from each service to have an influence on either workflow structure or other services in a workflow.

R2: Separation of Duty (SoD)
There are no obvious mechanisms for the language to be able to accommodate SoD. Conflict of interest can happen without any detection as services can be selected based on only cost optimization.

R3: The Association between Tasks and Services
Since time Petri Nets address both Atomic tasks and Atomic activities (services), this is the evidence to express association between tasks and services.

Although the language is able to specify which services are suitable for each task, in the aspect of flow-oriented execution, there are no mechanisms to express the attributes of services along the flow.

R5: Enforcement of Sequences
Time Petri Nets allow the enforcement of sequence, which strictly enforce the structure of flow execution. However, there are no mechanisms for services or workflow owners to manifest their requirements regarding the sequence of execution.

R6: Flexible Degrees of Restriction
Using static time Petri Nets to model a workflow is not sufficient to accommodate flexible degrees of restriction.

R7: Protection of Workflow Data
Data can be partly protected by the language, as there are strict requirements in place that screen potential services to be part of a workflow. However, how requirements are specified is not clearly stated.

3.2. Business Application Modeler (BAM) (Feja et al 2011)

BAM is a tool-based approach for requirements validation and verification for business process models. The core element is the formal specification that can be used for automatic validation and verification of process models. The requirements are specified as graphical rules in Computation Tree Logic (CTL), called Graphical Computational Tree Logic G-CTL (Feja and Fotsch 2008). Basic operators from CTL is employed including, EX, AX, EG, AF, EF, E(A U B), and A(A U B). Figure 2 depicts the symbols of G-CTL, being straightforward representations of the basic CTL operators.

Analysis:
As suggested in (Viriyasitavat 2013), the operators in CTL are insufficiently expressive to address requirements in service workflows. One major limitation is the lack of mechanisms to specify attribute composition of composed services in a task. For example, in a hotel rating task as a part of travel-agent workflow, one requirement states that at least two different services composed for this task must be presented. One is a certificate endorsed by HOTREC (Hotels, Restaurants & Cafés in Europe) and another is a certificate endorsed by HSU (Hotel Star Union). The language is unable to address this requirement. The analysis of this language according to principles in section 2 is as follows (see Table 1):
R1: Interoperability with Local Requirements
The language lacks mechanisms to address local requirements from each service to be part of service workflow.

R2: Separation of Duty (SoD)
CTL-based workflow specification language allows SoD requirement to be specified. However, it can only be applied to SoD based on the viewpoint of task, but not services executing a task.

R3: The Association between Tasks and Services
The language is unable to specify attributes of services to be part of a workflow.

Since there is no specification of attributes of services, task- and flow-oriented attributes of composed services cannot be addressed by the language.

R5: Enforcement of Sequences
However, in the aspect of enforcement of sequence, the language is expressive enough to specify the task sequence in a workflow path by using CTL operators.

R6: Flexible Degrees of Restriction
CTL by itself enables flexible degree of restriction, such as operators EF and AF.

R7: Protection of Workflow Data

---

**TABLE 1**
Comparison between Existing Works on Trust Formalism and Requirements of Specification Languages

<table>
<thead>
<tr>
<th>Models</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hsieh and Lin</td>
<td>×</td>
<td>×</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>√</td>
</tr>
<tr>
<td>BAM</td>
<td>×</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Davulcu et al</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>SOLOIST</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>SWSpec</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

✓ Direct Support, √/× Partial Support, × Not Support

---

**Fig. 2.** The symbols of G-CTL
Atomic proposition in CTL is able to specify how data security must be satisfied. Since the language is primarily based on CTL, the protection of workflow data is satisfied.

3.3. Concurrent Transaction Logic (CTR) (Davulcu et al 1999)

Davulcu et al. proposed a formal framework based on Concurrent Transaction Logic (CTR) for interaction modeling in virtual enterprises. Workflows are graphically expressed by a Direct Acyclic Graph (DAG), representing task coordination. A set of CTR connectives is invented to enforce constraints on a workflow, or specifically flow control. Like propositional logic, CTR formulae assume and verify truth-values specific to paths of a workflow. For example, if a formula is true over a path $<s_1, ..., s_n>$, the verification of that formula is true in the execution starting from $s_1$ to $s_n$. There are five important CTR connectives in this framework:

1. $\phi \otimes \psi$ means $\phi$ must precede $\psi$;
2. $\phi \mid \psi$ describes $\phi$ and $\psi$ are executed in parallel;
3. $\phi \land \psi$ indicates $\phi$ and $\psi$ are on the same path;
4. $\phi \lor \psi$ restricts $\phi$ or $\psi$ is executed; and
5. $\neg \phi$ represents the negation of the formula $\phi$.

Since the framework is rooted in logic, one major advantage is to allow automated reasoning about the intended behavior of virtual enterprises to ensure that they function as specified.

Analysis:

Even though these operators are sufficient to specify workflow constraints based on tasks coordination requirements, they are not expressive enough according principles described in section 2 (see Table 1).

R1: Interoperability with Local Requirements:
The language focuses on the flow control of a workflow. It does not accommodate requirements of services to be part of the flow structure.

R2: Separation of Duty (SoD)
Since the language ignores the involvement of services (task executors), SoD requirement cannot be addressed.

R3: The Association between Tasks and Services
As mentioned, the focus is on workflow flow control, the association between tasks and services are not manifested.

CTR is invented for the control flow, such that it can only specify in the aspect of flow-oriented specification.

R5: Enforcement of Sequences
CTR operator ($\otimes$) obviously enforces of the sequence of tasks in a workflow.

R6: Flexible Degrees of Restriction
None of CTR operators are capable of satisfying the degrees of restriction in specifying requirements.

R7: Protection of Workflow Data
Workflow data cannot be protected, since CTR is invented solely for the flow control of a workflow.
In addition, several other notions are required for this language in order to address workflow characteristics. For example, as a workflow is a tree-like structure, path quantifier operators are very much important.

### 3.4. Specification Language for service compositions inTeractions (SOLOIST) (Brianulli et al. 2013)

By following the Service Oriented Architecture (SOA) principles, Brianulli et al. developed SOLOIST to facilitate service-based application development by orchestrating third-party services to provide added-value applications. This service-based application, often defined as service compositions, has seen a wide adoption in enterprises, which nowadays develop their information systems using the principles of service orientation (Josuttis, 2007). The ownership of services in modern-age software system setting is distributed, in which changes are frequent, unexpected, and welcome (Papazoglou, 2008). Because of this evolving nature, service selection requires strong verification for overall correctness, dependability and quality of service attributes. The authors suggested that formal verification and expressive specification are two crucial elements to support automated verification. SOLOIST is a specification language for formalizing the interactions of service compositions. The language is based on first-order and temporal logics with extension to support aggregate operators for events occurring in a certain time window. SOLOIST is invented based on temporal logic with new modalities as follows. Let $t_1, ..., t_n$ are terms and $p(t_1, ..., t_n)$ is a formula and $\phi$ and $\psi$ are two formulae,

- $\phi U_I \psi$ and $\phi S_I \psi$: The $S_I$ and $U_I$ operators have the usual meaning in temporal logics (“Since” and “Until”) where $I$ is nonempty interval ($I \in \mathbb{N}$).
- $C^K_{\text{men}}(\phi)$: This operator, evaluated in a certain time instant, states a bound on the number of occurrences of an event $\phi$, counted over a time window $K$.
- $V^K_{\text{men}}(\phi)$: This operator, evaluated at a certain time instant $t_I$, is used to express a bound on the average number (with respect to an observation interval $h$ open to left and closed to the right) of occurrences of an event $\phi$, occurred within a time window $K$.
- $M^K_{\text{men}}(\phi)$: This operator, evaluated in a certain time instant $t_I$, is used to express a bound on the maximum number (with respect to an observation interval $h$, open to left and closed to the right) of occurrences of an event $\phi$, occurred within a time window $K$.

The rest of operators are defined in first-order and LTL operators.

**Analysis:**

The analysis of SOLOIST against principles described in section 2 is as follows (see Table 1):

**R1: Interoperability with Local Requirements**

Although terms in a predicate symbol $p(t_1, ..., t_n)$ can address all type of requirements to be expressed in SOLOIS, clear explanation and mechanisms to integrate local requirements from each service is not stated.

**R2: Separation of Duty (SoD)**

Since SOLOIST focuses on selecting one single service at a time for each task, SoD requirements, which usually deal with two or more services with conflict of interest, cannot be specified.

**R3: The Association between Tasks and Services**

SOLOIST formulae express requirements (attributes) of services being qualified for a task where different tasks can have different requirements. Therefore, the association between tasks and services is fully captured.

Although SOLOIST is able to specify requirements of services for each task, in the aspect of flow-oriented of execution, there are no mechanisms to express attributes of services along the flow. The reason is that SOLOIST is built based on LTL which fails to quantify over paths and branches of service workflows.

**R5: Enforcement of Sequences**

SOLOIST employs LTL operators such as “Next”, “Since”, and “Until”. Therefore, the sequence of events can be enforced. Furthermore, SOLOIST operators, $C^K_{on}$ and $M^K_{on}$, are even more expressive by including time to be part of the enforcement.

**R6: Flexible Degrees of Restriction**

SOLOIST operator $V^K_{on}(\phi)$, which deals with average number of occurrence, and $M^K_{on}$, which deals with the maximum number of occurrence, provide flexible degree of restriction. These operators can be used to specify requirements even the lack of the whole workflow visibility.

![Fig. 3. Example of required attributes p, q, and r for task 1 execution](image)

**R7: Protection of Workflow Data**

Data can be protected by SOLOIST by specifying requirements in terms $(t_i)$ inside a predicate symbol $p(t_1, ..., t_n)$, which can address, for example, how data is encrypted, which entities is restricted to provide data, and how many copies of data to be stored.

![Fig. 4. Example of The Travel Agent Services Workflow](image)

SWSpec language was invented to mathematically specify requirements in the context of service workflows from both workflow owners and services (Viriyasitavat 2016). The main contribution is to allow requirements to be independently and formally specified, subject to the willingness of services to participate in a workflow, and the workflow owners to regulate his workflows. Since SWSpec is based on formal language, the compliance checking can be automated.

One important capability of the language is to capture the structural characteristic of service workflows that consists of several tasks, where each task can be executed by several services. The coordination of these tasks forms paths and branches of a workflow. Figure 3 shows an example of service workflow. Shaded rectangle represents tasks and circle represents services. Three services with attributes p, q, and r are part of task 1. A workflow owner enforces requirements for service selection decision through SWSpec language that indicates attributes of services needed for each task. In the aspect of involved services, they can utilize SWSpec to specify their requirements regarding attributes of other services collaborated in the same workflow. Figure 4 illustrates more complex service workflow of a travel-agent business (Viriyasitavat 2013).

SWSpec algebra is defined in three types. (1) Composite formulae specify attribute composition for a task, (2) Path formulae quantify paths and branches of a workflow, and (3) Direction formulae indicate the direction to which the formulae applied. SWSpec grammar is shown as follows.
1. Composite Formulae

\[ S ::= !S \mid \mathcal{F} Z \mid \mathcal{P} Z \mid \mathcal{F} \mathcal{A} Z \mid \mathcal{P} \mathcal{A} Z \mid (S \& S) \mid (S \mid S) \quad \text{(Quantifier part)} \]
\[ Z ::= \varepsilon \mid !Z \mid (Z \cap Z) \mid (Z \cup Z) \mid (Z \oplus Z) \quad \text{(Property part)} \]

2. Path Formulae

\[ R ::= \top \mid \bot \mid S \mid \sim R \mid (R \land R) \mid (R \lor R) \mid \exists_t \ominus R \mid \exists_t \oslash R \mid \exists_t [R \cup R] \mid \forall_t \ominus R \mid \forall_t \oslash R \mid \forall_t [R \cup R] \mid \forall_t [R \cup R] \]

3. Direction Formulae

\[ W ::= \top \mid \bot \mid \mathcal{H} R \mid \mathcal{B} R \mid \sim W \mid (W \lor W) \mid (W \lor W) \]

Composite operators consist of quantifier parts and property part. The quantifier part is used to quantify services associated with and the property part is used to specify the attributes of service for that task. All path operators are similar to CTL to capture branches and sequence of tasks. Finally direction operators are used to...
specify forward and backward directions along a workflow path. The meaning of each operator is given in table 2.

Analysis:

R1: Interoperability with Local Requirements
SWSpec is specifically invented to address the requirements form both workflow owners and involved services. It allows each service to express its own requirements to other services in a workflow. Therefore, local requirements from each involved services are part of workflow verification, enabling interoperability with local requirements

R2: Separation of Duty (SoD)
SoD specific requirement can be addressed by SWSpec. For example, in a financial audit scenario, the annual financial statement must be audited by two different auditing companies. In this case, it is not necessary to precisely identify the specific companies, but instead have to confirm that they are from different companies. The atomic proposition can be extended by introducing a dummy variable $d_i$ as $\exists P_{\text{Audit}}((d_i, \varepsilon, \{\text{Audit}\}) \cap (\neg d_i, \varepsilon, \{\text{Audit}\})$. This statement means that two different services must be present to execute the audit task (Viriyasitavat and Martin 2011b).

R3: The Association between Tasks and Services
Composite formulae are responsible for task and services involved for that task. The association between tasks and services are clearly addressed by SWSpec.

Since SWSpec Path formulae are based on CTL operators, flow oriented attribute specification can directly be addressed. SWSpec Composite formulae are used for specification of service composition for a task. This captures the requirement in the aspect of task-oriented attribute specification.

R5: Enforcement of Sequence
NEXT operator in CTL can obviously enforce of the sequence of tasks in a workflow.

R6: Flexible Degrees of Restriction
As already mentioned, CTL by itself enables flexible degree of restriction, such as EF and AF.

R7: Protection of Workflow Data:
Atomic proposition in SWSpec can be extended to address any type of security requirements. For instance, the data flown from one task to another in a service workflow can be protected by specifying all services along the flow must encrypt its results.

4. Conclusion

To achieve workflow interoperability in dynamic environments, formal requirements specification is regarded as an effective tool to solve interoperation and service selection problems. A number of services may offer same functionality with different attributes. Dynamicity in real-time service workflow interoperation creates complexity, in which changes of services’ attributes can be constantly occurred (Viriyasitavat 2016). This requires formal language to specify attributes of services, being qualified for workflow participation. This paper provides the reviews and analysis of the state-of-the-art service workflow specification languages, including (1) Self-Adaptive Configuration based on HMS (2) BAM, (3) CTR, (4) SOLOIST, and (5) SWSpec. These languages are evaluated against the fundamental principles needed for general service workflow specification languages. The review suggests that SWSpec is the only language that addresses all aspects of the principles.
References


15


