

Time-Constrained, Event-Driven Coordination of Composite Resources' Consumption Flows

Zakaria Maamar¹ ^a, Amel Benna² ^b and Nabil Otsmane³

¹College of Computing and IT, University of Doha for Science and Technology, Doha, Qatar

²Department of Multimedia and Information Systems, CERIST, Algiers, Algeria

³High School of Computer Science, Algiers, Algeria

Keywords: Allen's Time Algebra, Composite Resource, Consumption Flow, Event Driven, Time Constraint.

Abstract: This paper discusses the composition of primitive resources in preparation for their run-time consumption by business processes. This consumption is first, subject to time constraints impacting the availability of primitive resources and second, dependent on events impacting the selection of primitive resources. To address primitive resources' disparate time-availabilities that could lead to conflicts, a coordination approach is designed, developed, and tested using Allen's time algebra and a simulated dataset. The approach produces composite resources' consumption flows on-the-fly after discovering time relations between primitive resources that ensure their availabilities and hence, assignment to business processes. Implementation results demonstrate the technical doability of the approach along with identifying time-related obstacles that could prevent primitive resources' availabilities. Solutions addressing these obstacles are also reported in the implementation results.


1 INTRODUCTION


In (Maamar and Al Khafajiy, 2021) and (Maamar et al., 2021), we discuss the concepts of resource, consumption of resource, consumption property, consumption cycle, and disruption, and exemplify all these concepts with cloud computing. The discussions demonstrated the importance of a coordinated consumption of resources in a multi-job environment since the continued availability of resources is not always guaranteed (Dimick, 2014); some are scarce while others decline. Example of resource would be virtual machine, example of consumption property would be limited-but-extensible, example of disruption would be urgent upgrade of a virtual machine, and, last but not least, example of job would be order delivery. Assumption made in (Maamar and Al Khafajiy, 2021) and (Maamar et al., 2021) is that a job would consume one resource at a time, which is not always the case in real life. A job like order delivery, again, would consume processing power/CPU to run programs, storage capacity/database to store data, and bandwidth/router to communicate data. By dropping this assumption in the sense that a job would now

consume many resources, this means specializing resource into primitive and composite.

By analogy with tasks and component Web services that are put together to form business processes (Weske, 2012) and composite Web services (Langdon, 2003), respectively, a composite resource would line up a set of primitive, and even composite, resources that would be consumed according to specific events that would arise and specific functional and non-functional characteristics that jobs would have. However, lining up several primitive resources would require considering their respective availability times. For instance, a primitive resource is available between 2pm and 4pm, only, while another is available after 10min from the complete consumption of a group of primitive resources. How to coordinate the consumption of composite resources according to their primitive resources' availability times is a concern that we address in this paper.

In conjunction with achieving a time-constrained, event-driven coordination of composite resources, the impact of primitive resources' consumption properties on composite resources' availability times could hinder this coordination. Specialized into unlimited, limited, limited-but-extensible, shareable, and non-shareable as per our work in (Maamar et al., 2016), consumption properties could for instance, impose

^a  <https://orcid.org/0000-0003-4462-8337>

^b  <https://orcid.org/0000-0002-9076-5001>

extending the availability times of some primitive resources at the expense of others so that the consumption of the composite resource as a whole completes successfully. How to perform this extension without initiating conflicts is another concern that we address in this paper, as well.

To address both concerns, we resort to Allen's time algebra (Allen, 1983). The objective is to identify potential time-interval relations between the respective availability times of primitive resources participating in the same composite resource. Allen's algebra offers an exhaustive coverage of possible relations between time intervals along with the possibility of reasoning over these relations. Examples of relations include equals, overlaps, starts, and during. Our objective is to achieve a time-constrained, event-driven coordination of composite resources that would be sensitive to both availability times of and consumption properties of primitive resources. Our contributions are, but not limited to, (i) identification of potential time-interval relations between primitive resources' availability times using Allen's time algebra, (ii) analysis of the impact of consumption properties on primitive resources' availability times, (iii) on-the-fly definition of consumption flows of composite resources based on their primitive resources' availability times and consumption properties, and (iv) demonstration of consumption flows through a case study and system. The rest of this paper is organized as follows. Section 2 is a summary of some related works. Section 3 defines the concepts of resource and their consumption properties and also refers to a running example used for illustration purposes. Section 4 provides a temporal analysis of these consumption properties prior to detailing the approach for coordinating the consumption of primitive resources in Section 5. This approach's technical details and concluding remarks are reported in Sections 6 and 7, respectively.

2 RELATED WORK

In the research community, resource management in business processes is commonly studied. In this section, we discuss this management in terms of resource allocation and composition. In (Stefanini et al., 2020), Stefanini et al. propose a process mining-based approach to support resource planning of health services. They combine techniques like time-driven activity-based costing and process mining to identify and analytically evaluate tasks, service times, and resource consumptions for specific medical conditions. For the needs of process mining, the approach uses an

event log to estimate the expected resource consumptions of each medical intervention.

In (Maamar et al., 2022), Maamar et al. define an approach for coordinating the consumption of resources by business processes' tasks. The approach takes into account both resources' properties like unlimited and limited-but-extensible and tasks' transactional properties like pivot and compensatable. On top of these properties, the approach adopts Allen's time algebra and uses historical details about past executions stored in an event log to coordinate resource consumption. In (Arias et al., 2015), Arias et al. propose a process mining-based recommendation framework to allocate resources to sub-processes instead of individual tasks. The framework uses a set of criteria related to resource's capabilities, resource's workload, necessary expertise to perform tasks, and event log that encompasses details about previous executions. Mixing these criteria allowed recommending the top-ranked resources to a sub-process based on the best position algorithm (Akbarinia et al., 2011).

In (Park and Song, 2019), Park and Song build upon the results of predictive process monitoring to improve business processes especially resource allocation. To optimize this allocation, the authors use Long Short-Term Memory (LSTM) to predict the processing time of each task and next task of an ongoing process instance. These details (i.e., processing time and next task) are exploited to allocate resources to tasks thanks to a minimum cost and maximum flow algorithm. In (Sindhgatta et al., 2016), the authors propose a context-based approach for making decisions about resource allocation to tasks. The approach relies on historical data, process context, and performance of past instances all stored in an event log to predict the performance of under-execution process instances. Resources allocated to these instances are taken care by k-Nearest Neighbor technique.

In (Zhao et al., 2016), Zhao et al. analyze resource allocation to a business process's tasks as a multi-criteria decision making problem. The recommendation of resources to a BP's tasks considers both resource characteristics and task preference patterns established based on past executions. To uncover these patterns, the authors adopt an entropy-based clustering ensemble method. In addition, they provide dynamic resource allocation for concurrently running process instances. In a recent work (Zhao et al., 2020), Zhao et al. address the problem of human resources allocation using team faultlines. First, resources' characteristics are described from demographic and past execution perspectives. Then, this faultline is identified and measured using various characteristics and multiple subgroups. Finally, a

neural network is used to achieve the allocation.

The paragraphs above reveal that resource allocation problem in business processes is timely based on the different techniques that aim at optimizing this allocation. Our approach is in line with what the research community's efforts with focus on resource composition according to these resources' time availabilities and consumption properties. To synchronize these availabilities along with these properties, Allen's time algebra is used permitting to form what we refer to as consumption flows.

3 BACKGROUND

This section defines the concept of resource, presents a running example, and finally discusses consumption properties of resources and Allen's time Algebra.

Resource Definition. The concept of resource is not new in the literature and has been adopted in different domains like distributed artificial intelligence (e.g., resource logic for multi-agent planning (de Weerd and Clement, 2009)), service computing (e.g., RDF for interoperability (Schreiber and Raimond, 2014) and REST for building applications (Web, Intranet, and Web services) (Fielding, 2000)), cloud computing (abstracting hardware and software (Mell and Grance, 2011)), and business processes (persons/machines executing tasks (Fielding, 2000)). From a broad perspective, Baker et al. specialize resources into computational, consumed, and produced, associating each type with a separate lifecycle that would capture the resource's behavioral and operational characteristics (Baker et al., 2018). In (Lucchim et al., 2008), Lucchim et al. consider resources as "directly-accessible components handled through a standard common interface". The interface could be a set of stateless operations like HTTP methods. Finally, Hofman adopts everything-as-a-resource to build seamless interoperable platforms in the world of IoT (Hofman, 2015). Each resource (e.g., truck and smart object) has goals and capabilities and may have an owner, user, and virtual representation.

Running Example. It is about John who is visiting Melissa in Paris. One day they decide to meet in a coffee shop, not far from Melissa's office. John can reach the coffee shop by either taxi or bus. We adopt Web services consuming on-premise and in-the-cloud resources to suggest a high-level description of the job completing John scenario (Fig. 1). At the hotel, John browses some transportation Web sites about Paris. One running Itinerary WS and hence, consuming processing power, proposes routes between places. While checking the weather fore-

cast using Weather WS, Itinerary WS requests details about the origin and destination places using Location WS that accesses a dedicated database. Should Weather WS return bad weather, Itinerary WS would instruct Taxi WS to book a taxi for John. Otherwise, Itinerary WS would send the location of both John's hotel and the coffee shop to Bus WS, which advises about the bus numbers that John should ride. Potential traffic jams would make Bus WS interact with Traffic WS, should the bus numbers need to be adjusted.

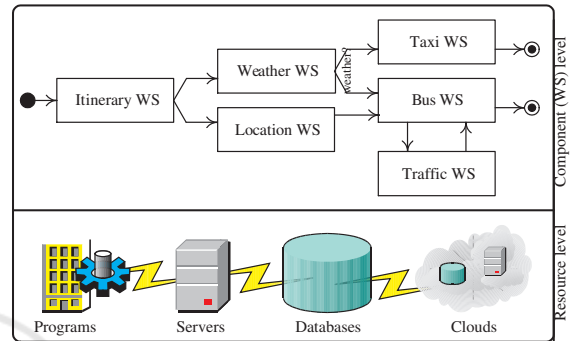


Figure 1: Specification of the job handling John scenario.

From a consumption perspective, on-premise and in-the-cloud resources would be associated with necessary consumption properties as their owners see fit. For instance, databases could be non-shareable for concurrent Web services so that consistency is enforced, and servers hosting virtual machines could be limited-but-extensible for long-running Web services so that a balanced load over the servers is maintained. Besides the consumption properties, the resources could have availability times that need to be considered, too. For instance, backup databases are activated concurrently everyday after midnight for 3 hours, and servers hosting virtual machines are suspended sequentially once a week for 2 hours for maintenance. Both consumption properties and availability times of resources would have an impact on completing the job above.

Consumption Properties of Primitive Resources. In compliance with our previous work on social coordination of business processes (Maamar et al., 2016), the consumption properties of a primitive resource (\mathcal{PR}) could be unlimited (u), shareable (s), limited (l), limited-but-extensible (lx), and non-shareable (ns). First, a primitive resource is limited when its consumption is bound to an agreed-upon time period (capacity, too, like 20 liters, but not considered in this work). Second, a primitive resource is limited-but-extensible when its consumption continues to happen after extending the (initial) agreed-upon time period. Finally, a primitive resource is non-

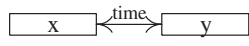
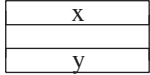
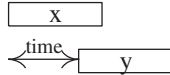
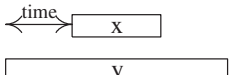
shareable when its concurrent consumption needs to be coordinated (e.g., one at a time). A primitive resource is by default unlimited and/or shareable. The set of all consumption cycles (CC) of the 5 properties are captured into Fig. 2. However, only 2 consumption cycles are listed for illustration purposes.

1. Unlimited property: $\mathcal{PR}.cc_{ul}$: not-made-available $\xrightarrow{\text{start}}$ made available $\xrightarrow{\text{waiting-to-be-bound}}$ not-consumed $\xrightarrow{\text{consumption-approval}}$ consumed $\xrightarrow{\text{no-longer-useful}}$ withdrawn.
2. Limited-but-extensible property: $\mathcal{PR}.cc_{lx}$: not-made-available $\xrightarrow{\text{start}}$ made available $\xrightarrow{\text{waiting-to-be-bound}}$ not-consumed $\xrightarrow{\text{consumption-approval}}$ consumed $\xrightarrow{\text{renewable-approval}}$ made available. The transition from done to made available allows a resource to be regenerated for another cycle of consumption.

Definition 1. A primitive resource \mathcal{PR} is defined by the tuple $\langle id, name, period, cp, cc_{cp} \rangle$ where id is the identifier of the primitive resource, $name$ is the name of the primitive resource, $period$ is the availability-time interval $[b, e]$ of the primitive resource where b and e stand for begin-time and end-time, cp is the consumption property of the primitive resource as either $u, s, l, lx,$ or ns , and cc_{cp} is the consumption cycle of the primitive resource according to its consumption property and is defined by the tuple $\langle S, T, S_{active} \rangle$, i.e., $cc_{cp} = s_i \xrightarrow{\text{trans}_i} s_{i+1} \xrightarrow{\text{trans}_{i+1}} s_{i+2} \dots s_{j-1} \xrightarrow{\text{trans}_{j-1}} s_j$ where S is the set of states $\{s_i\}$ where s_i is either not-made-available, consumed, locked, etc. (Fig. 2), T is the set of transitions $\{\text{trans}_i\}$ where trans_i is either consumption-approval, consumption-update, etc. (Fig. 2), and $S_{active} \subset S$ is the set of states that have been enabled since the activation of the consumption cycle. At initialization time, $S_{active} = \{\text{not-made-available}\}$.

Allen's time algebra. Table 1 presents some potential relations (in fact, there exist 13) between time intervals, i.e., pairs of endpoints, allowing to support multiple forms of temporal reasoning like what to do when 2 time intervals start/end together, when a time interval falls into another time interval, etc. (Allen, 1983). In Allen's time algebra, each relation is labeled as either distinctive, exhaustive, or qualitative. Typical applications of Allen's time algebra include planning and scheduling, natural language processing, temporal databases, workflows, to cite just some (Janhunen and Sioutis, 2019).

Table 1: Some Allen's time-interval relations.

	
$x \text{ precedes } y$	$x \text{ equals } y$
	
$x \text{ overlaps } y$	$x \text{ during } y$

4 TEMPORAL ANALYSIS OF CONSUMPTION PROPERTIES

When tracking the consumption of a primitive resource (\mathcal{PR}_k) by a job (J_i) from a time-interval perspective, this consumption, that could happen many times, i.e., $1[con_{in=1,\dots}]^*$, would depend on the primitive resource's both consumption property and availability-time interval and would allow to define what we refer to as the job's consumption-time intervals (when the consumption has effectively occurred). To track how a job consumes a primitive resource, we proceed as follows:

1. Unlimited, limited, and limited-but-extensible primitive resources, we associate them with specific availability-time intervals, $\mathcal{PR}_k[b, \infty[$, $\mathcal{PR}_k[b, e]$, $\mathcal{PR}_k[b, e 0[+\delta]^*$, where b , e , and δ stand for begin-time, end-time, and extra-time (repeated but not indefinitely), respectively.
2. Shareable and non-shareable primitive resources, we either tolerate or not their concurrent consumption ($con_{in}, con_{jn'}, \dots$) by separate jobs (J_i, J_j, \dots) during the respective availability-time intervals of these primitive resources, e.g., $J_i^{\mathcal{PR}_k}[b_{con_{in}}, e_{con_{in}}] \subseteq \mathcal{PR}_k[b, e] \wedge J_j^{\mathcal{PR}_k}[b_{con_{jn'}}, e_{con_{jn'}}] \subseteq \mathcal{PR}_k[b, e]$ and $b_{con_{in}} == b_{con_{jn'}}$.
3. Unlimited primitive resources, we allow them to accommodate any job's multiple consumption requests ($con_{i1}, con_{i2}, \dots$) during their availability-time intervals.

To illustrate the 3 cases above, Table 2 refers to 3 jobs, J_1 , J_2 , and J_3 , and their respective consumption of resources for instance, J_1 's con_{11} , J_2 's $con_{21,22}$, and J_3 's $con_{31,32,33,34}$. We now discuss the impact of limited and limited-but-extensible consumption properties on a primitive resource's availability-time interval using the same table.

1. Limited property means that a primitive resource's availability-time interval that is set

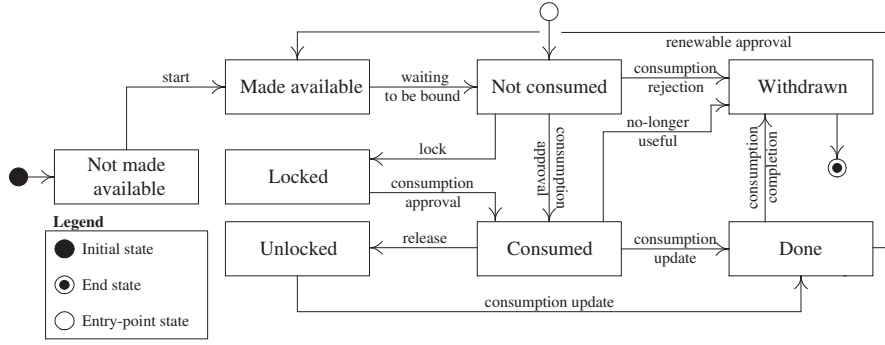
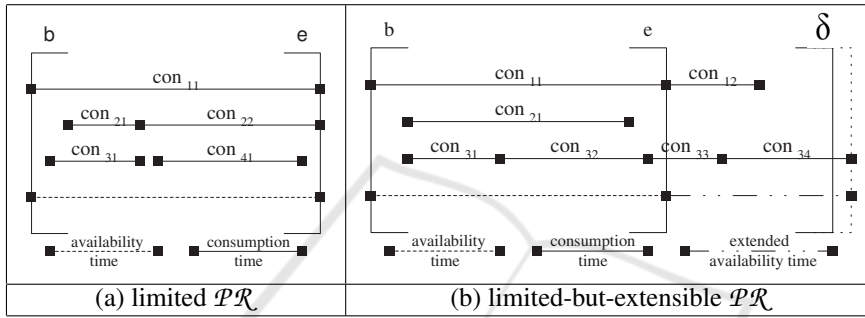


Figure 2: Primitive resource's consumption cycle as a state diagram (Maamar et al., 2016).

 Table 2: Availability-time interval *versus* Consumption-time interval.


at design-time, $\mathcal{PR}_k[b, e]$, remains the same at run-time despite the requests for additional consumption that this primitive resource would receive from the same jobs (after these jobs' respective first consumption). A job requesting to consume a limited primitive-resource is confirmed *iff* the job's first consumption-time interval falls into the primitive resource's availability-time interval (e.g., $\mathcal{J}_2^{\mathcal{PR}_k}[b_{con_{21}}, e_{con_{21}}] \subset \mathcal{PR}_k[b, e]$ in Table 2 (a) where $b_{con_{21}} > b$ and $e_{con_{21}} < e$) and, then, any additional consumption-time intervals must fall into the primitive resource's same availability-time interval (e.g., $\mathcal{J}_2^{\mathcal{PR}_k}[b_{con_{22}}, e_{con_{22}}] \subseteq r_k[b, e]$ in Table 2 (a) where $b_{con_{22}} = e_{con_{21}}$ and $e_{con_{22}} = e$).

2. Limited-but-extensible property means that a primitive resource's availability-time interval that is set at design-time, $\mathcal{PR}_k[b, e]$, can be adjusted at run-time, $\mathcal{PR}_k[b, e + \delta]$, so that the requests for additional consumption that this primitive resource would receive from the same jobs (after these jobs' respective first consumption) are accommodated. A job requesting to consume a limited-but-extensible primitive-resource is confirmed *iff* the job's first consumption-time interval falls into the primitive resource's availability-time interval (e.g., $\mathcal{J}_3^{\mathcal{PR}_k}[b_{con_{31}}, e_{con_{31}}] \subset \mathcal{PR}_k[b, e]$

in Table 2 (b) where $b_{con_{31}} > b$ and $e_{con_{31}} < e$) and, then, any additional consumption-time intervals still fall into either the primitive resource's same availability-time interval (e.g., $\mathcal{J}_3^{\mathcal{PR}_k}[b_{con_{32}}, e_{con_{32}}] \subset \mathcal{PR}_k[b, e]$ in Table 2 (b) where $b_{con_{32}} = e_{con_{31}}$ and $e_{con_{32}} < e$) or the primitive resource's extended availability-time interval (e.g., $\mathcal{J}_3^{\mathcal{PR}_k}[b_{con_{33}}, e_{con_{33}}] \subset \mathcal{PR}_k[e, e + \delta]$ in Table 2 (b) where $b_{con_{33}} = e_{con_{32}}$ and $e_{con_{33}} < e + \delta$).

5 CONSUMPTION OF COMPOSITE RESOURCES

After defining some concepts linked to the coordinated consumption of composite resources, we now discuss how these resources' consumption flows are generated according to primitive resources' availability times and consumption properties.

5.1 Definitions

In preparation for defining composite resources, we deem necessary contrasting them with business processes and composite Web services. On the one hand, while a business process and composite Web service

have a process model and composition schema, respectively, a composite resource also has a consumption flow. In addition, while the performance of a business process and composite Web service is dependent on their tasks' and component Web services' non-functional properties, respectively, the consumption of a composite resource is also dependent on their primitive resources' availability-time intervals and consumption properties. On the other hand, while a business process' process model and composite Web service's composition schema are known ahead of time, a composite resource's consumption flow is set on-the-fly according to their primitive resources' availability-time intervals and consumption properties and Allen's time-interval relations that could vary from one consumption cycle to another (begin and end values assigned to time intervals vary per scenario). Finally, the completion of business processes and composite Web services depends on the availability of composite resources that themselves depend on the availability of primitive resources.

Definition 2. A composite resource $C\mathcal{R}$ is defined by the tuple $\langle id, name, evt, \{UPR\}, C\mathcal{F} \rangle$ where id is the identifier of the composite resource, $name$ is the name of the composite resource, evt is the name of the event that triggers the consumption of the composite resource, UPR is an unordered set of primitive resources $\{\mathcal{P}\mathcal{R}_i\}$ as per Definition 1, and $C\mathcal{F}$ is the consumption flow of the composite resource that is generated on-the-fly based on both the primitive resources' availability-time intervals/consumption properties and Allen's appropriate time-interval relations between these primitive resources. The consumption flow is represented as a set of couples $\{rel(\mathcal{P}\mathcal{R}_i, \mathcal{P}\mathcal{R}_j)\}$ where $rel \in \{equals, starts, finishes, overlaps, during\}$. More details are given in Section 5.2

5.2 Generation of Consumption Flows

Let us assume a job (\mathcal{J}) to carry out like in the running example. The generation of this job's composite resource's consumption flow ($C\mathcal{F}$) would go over *identification* and *adjustment* stages (Fig. 3). Briefly, in the identification stage, the designer defines Allen's time-interval relations that would connect the primitive resources together based on these primitive resources' respective availability-time intervals. This produces what we refer to as the composite resource's initial consumption flow ($IC\mathcal{F}$). And, in the adjustment stage, the designer assesses the impact of the identified Allen's time-interval relations on the primitive resources' consumption properties. This could require adjusting the availability-time intervals of these

primitive resources to enforce their concurrent/composite consumption whenever deemed possible. The assessment goes through several rounds until what we refer to as the composite resource's final consumption flow ($\mathcal{F}C\mathcal{F}$) is obtained.

Identification Stage. As per Definition 2, the unordered set of primitive resources (\mathcal{UPR}) that form a composite resource are known based on the triggered event of the job to complete. This set constitutes an input to Algorithm 1 whose output is the composite resource's $IC\mathcal{F}$. In $IC\mathcal{F}$, the different primitive resources are connected together through appropriate Allen's time-interval relations, e.g., $IC\mathcal{F} = \{overlaps(\mathcal{P}\mathcal{R}_i, \mathcal{P}\mathcal{R}_j), \dots, starts(\mathcal{P}\mathcal{R}_k, \mathcal{P}\mathcal{R}_l)\}$.

Adjustment Stage. The outcome of the identification stage that is $IC\mathcal{F}$ is processed according to the following steps:

Step 1: parses the $IC\mathcal{F}$ to cluster the primitive resources based on Allen's time-interval relations. The result are 5 clusters, $C_{rt=1..5}^{rd=1..m}$, where a cluster's superscript (rd)¹ and subscript (rt) correspond to the current number of round tracking the progress of parsing the $IC\mathcal{F}$ and a numerical value that is the time relation's type, respectively. For instance, C_1^1 is the cluster of all primitive resources connected through *equals* at round 1 and C_2^3 is the cluster of all primitive resources connected through *starts* at round 3. It is worth noting that some clusters could end-up empty depending on the time-relation intervals included in $IC\mathcal{F}$.

Step 2: flushes \mathcal{UPR} and generates a unique composite resource, $C\mathcal{R}_{ij}^{rd}$, for each Allen's time-interval relation included in the 5 clusters, $C_{rt=1..5}^{rd}$, that illustrate a concurrent consumption of primitive resources, namely *equals*, *starts*, *finishes*, *overlaps*, and *during*. For instance, $C\mathcal{R}_{12}^1$ is the composition of $\mathcal{P}\mathcal{R}_1$ and $\mathcal{P}\mathcal{R}_2$ at round 1. Prior to inserting $C\mathcal{R}_{ij}^{rd}$ into the flushed \mathcal{UPR} that will become an unordered set of composite resources, $C\mathcal{R}_{ij}^{rd}$'s availability-time interval needs to be defined first, according to both Allen's time-interval relation of the cluster and the consumption properties and availability-time intervals of the primitive resources in the cluster as well and second, in a way that the concurrent consumption of the primitive resources is maintained.

- *Equals*($\mathcal{P}\mathcal{R}_i, \mathcal{P}\mathcal{R}_j$): whether $\mathcal{P}\mathcal{R}_i$ and $\mathcal{P}\mathcal{R}_j$ are limited or limited-but-extensible, $C\mathcal{R}_{ij}^{rd}$'s availability-time interval is the availability-time interval of any primitive resource, for in-

¹The maximum value, m , is known after completing the adjustment stage.

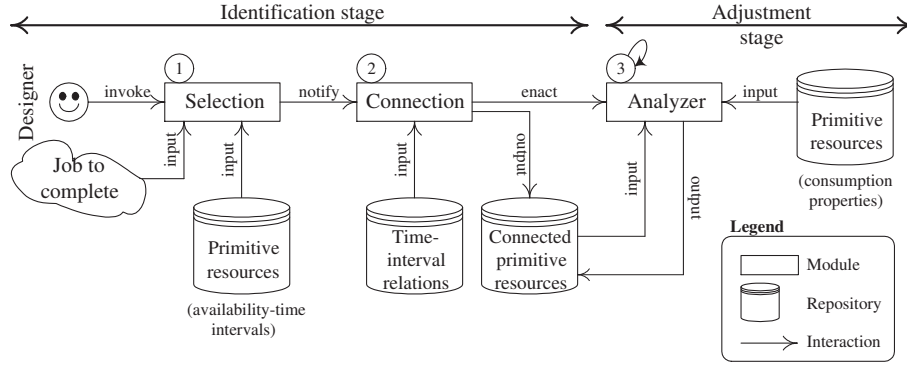


Figure 3: Modules, repositories, and interactions in the generation of consumption flows

- stance $[b_i, e_i]$.
- $Starts(\mathcal{PR}_i, \mathcal{PR}_j)$:
 - Limited property (Table 3-Case a): \mathcal{CR}_{ij}^{rd} 's availability-time interval is set as follows: begin-time of any primitive resource and end-time of \mathcal{PR}_i since it has the first end-time, for instance $[b_i, e_i]$.
 - Limited-but-extensible property (Table 3-Case b): \mathcal{CR}_{ij}^{rd} 's availability-time interval is set as follows: begin-time of any primitive resource and end-time of \mathcal{PR}_j that has the last end-time, for instance $[b_i, e_j]$. At the same time, \mathcal{PR}_i 's end-time is extended to match \mathcal{PR}_j 's end-time, $e_i + \delta == e_j$.
 - $Finishes(\mathcal{PR}_i, \mathcal{PR}_j)$:
 - Limited property: \mathcal{CR}_{ij}^{rd} 's availability-time interval is set as follows: begin-time of \mathcal{PR}_i that has the last begin-time and end-time of any primitive resource.
 - Limited-but-extensible property: \mathcal{CR}_{ij}^{rd} 's availability-time interval is set as follows: begin-time of \mathcal{PR}_i that has the last begin-time and end-time of any primitive resource.
 - $During(\mathcal{PR}_i, \mathcal{PR}_j)$:
 - Limited property: \mathcal{CR}_{ij}^{rd} 's availability-time interval is set as follows: begin-time of \mathcal{PR}_i that has the last begin-time and end-time of \mathcal{PR}_i again that has the first end-time.
 - Limited-but-extensible property: \mathcal{CR}_{ij}^{rd} 's availability-time interval is set as follows: begin-time of \mathcal{PR}_i that has the last begin-time and end-time of \mathcal{PR}_j that has the last end-time. At the same time, \mathcal{PR}_i 's end-time is extended to match \mathcal{PR}_j 's end-time, $e_i + \delta == e_j$.
 - $Overlaps(\mathcal{PR}_i, \mathcal{PR}_j)$:
 - Limited property: \mathcal{CR}_{ij}^{rd} 's availability-time interval is set as follows: begin-time of \mathcal{PR}_j

that has the last begin-time and end-time of \mathcal{PR}_i that has the first end-time.

- Limited-but-extensible property: \mathcal{CR}_{ij}^{rd} 's availability-time interval is set as follows: begin-time of \mathcal{PR}_j that has the last begin-time and end-time of \mathcal{PR}_j again that has the last end-time. At the same time, \mathcal{PR}_i 's end-time is extended to match \mathcal{PR}_j 's end-time, $e_i + \delta == e_j$.

Step 3: increments the number of round by 1, runs Algorithm 1 again with \mathcal{UPR} as an unordered set of composite resources, and goes back to Step 1 until the 4 clusters, except for *equals*, become empty, i.e., $C_{rt=2..5}^{rd} == \emptyset$. Should the condition hold for the 4 clusters, then the composite resource's \mathcal{FCF} is depicted in C_1^{rd} linked to *equals*, e.g., $\mathcal{FCF} = \{equals(\mathcal{CR}_i^{rd}, \mathcal{CR}_j^{rd}) \cup equals(\mathcal{CR}_k^{rd}, \mathcal{CR}_l^{rd}) \cup equals(\mathcal{CR}_m^{rd}, \mathcal{CR}_n^{rd}), \dots\}$. All the composite resources will have the same interval.

5.3 Illustration

With respect to the running example, the Web services consume different resources like virtual machines for processing and databases for storage. To illustrate the generation of a consumption flow, we assume an \mathcal{UPR} consisting of 4 primitive resources, $\mathcal{PR}_{1..4}$, all limited, having each an availability-time interval, for instance [2,5], [3,6], [2,5], and [3,5], respectively. We proceed with rolling out the identification and adjustment stages. After completing Algorithm 1, $\mathcal{ICF} = \{\overlaps(\mathcal{PR}_1, \mathcal{PR}_2), equals(\mathcal{PR}_1, \mathcal{PR}_3), \overlaps(\mathcal{PR}_3, \mathcal{PR}_2), starts(\mathcal{PR}_2, \mathcal{PR}_4), finishes(\mathcal{PR}_4, \mathcal{PR}_3), finishes(\mathcal{PR}_4, \mathcal{PR}_1)\}$. Then, the final interval would be [3,5].

Input: \mathcal{UPR} : $\{\mathcal{PR}_i[b_i, e_i]\}_{i=1..n}$: unordered set of primitive resources

Output: ICF : initial consumption flow of a composite resource

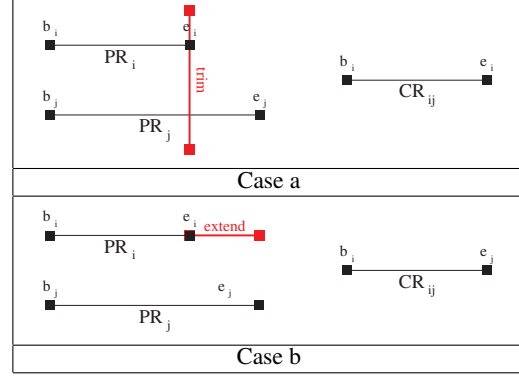
```

1 begin
2   Initialize  $ICF$  to  $\emptyset$ ;
3   Initialize Concurrency to true;
4   foreach  $i \in [1, n-1]$  do
5     foreach  $j \in [i+1, n]$  do
6       if Concurrency == true then
7         if  $e_i == b_j$  then
8            $ICF = \emptyset$ ; Concurrency  $\leftarrow$  false;
9           break;
10        end
11        if  $b_i == e_j$  then
12           $ICF = \emptyset$ ; Concurrency  $\leftarrow$  false;
13          break;
14        end
15        if  $b_i > b_j$  and  $e_j < b_i$  then
16           $ICF = \emptyset$ ; Concurrency  $\leftarrow$  false;
17          break;
18        end
19        if  $e_i > b_j$  and  $e_i < b_j$  then
20           $ICF = \emptyset$ ; Concurrency  $\leftarrow$  false;
21          break;
22        end
23        if  $(e_i == e_j \text{ and } b_i == b_j)$  then
24           $ICF = ICF \cup \text{equals}(\mathcal{PR}_i, \mathcal{PR}_j)$ ;
25          break;
26        end
27        if  $b_i == b_j$  and  $e_i > e_j$  then
28           $ICF = ICF \cup \text{starts}(\mathcal{PR}_i, \mathcal{PR}_j)$ ;
29          break;
30        end
31        if  $b_i == b_j$  and  $e_i < e_j$  then
32           $ICF = ICF \cup \text{starts}(\mathcal{PR}_i, \mathcal{PR}_j)$ ;
33          break;
34        end
35        if  $e_i == e_j$  and  $b_i < b_j$  then
36           $ICF = ICF \cup \text{finishes}(\mathcal{PR}_i, \mathcal{PR}_j)$ ;
37          break;
38        end
39        if  $e_i == e_j$  and  $b_i > b_j$  then
40           $ICF = ICF \cup \text{finishes}(\mathcal{PR}_i, \mathcal{PR}_j)$ ;
41          break;
42        end
43        if  $b_i < b_j$  and  $e_i < e_j$  then
44           $ICF = ICF \cup \text{during}(\mathcal{PR}_i, \mathcal{PR}_j)$ ;
45          break;
46        end
47        if  $b_i < b_j$  and  $e_j > b_i$  then
48           $ICF = ICF \cup \text{overlaps}(\mathcal{PR}_i, \mathcal{PR}_j)$ ;
49          break;
50        end
51        if  $e_i > b_j$  and  $e_i > e_j$  then
52           $ICF = ICF \cup \text{during}(\mathcal{PR}_i, \mathcal{PR}_j)$ ;
53          break;
54        end
55        if  $e_i > b_j$  and  $e_i < e_j$  then
56           $ICF = ICF \cup \text{overlaps}(\mathcal{PR}_i, \mathcal{PR}_j)$ ;
57          break;
58        end
59      end
60    end
61  end
62  return  $ICF$ 
63 end

```

Algorithm 1: Initial consumption-flow development.

Table 3: Definition of a composite resource's availability-time interval because of *starts*.



6 IMPLEMENTATION

This section discusses first the system implementing the approach for composing primitive resources and then, the experiments to test and evaluate this system.

6.1 Overview

To demonstrate the technical doability of on-the-fly development of consumption flows of composite resources, a system whose main graphical user interface is shown in Fig. 4, was implemented using multiple languages, tools, and technologies. This includes Java, NodeJS (a back-end JavaScript runtime environment), Spring boot (a Java-based open source framework to build and deploy Web applications), Gradle (a tool for automating software build), ReactJS (a front-end JavaScript library to build user interfaces), JSON (an open standard for data format and interchange), Visual Studio Code (an open source lightweight-code editor for JavaScript and TypeScript languages), and Postman (a platform for testing components represented as API endpoints). Classes in the system are primitiveResource, compositeResource, relation-Allen, and consumptionFlowGenerator implementing the model-view-presenter architectural pattern. The development and experimentation of the system took place on top of an 11th Generation Intel 4 Cores i5 Processors@4.2GHZ, 16GB RAM desktop.

To use the system, a designer stores necessary details about jobs to complete using primitive resources into JSON files and then, uploads these files into the system as per Fig. 4. In this figure, “All are concurrent”, “Part are concurrent”, and “Both” (not-considered further) indicate possible scenarios to test as per the next paragraphs.

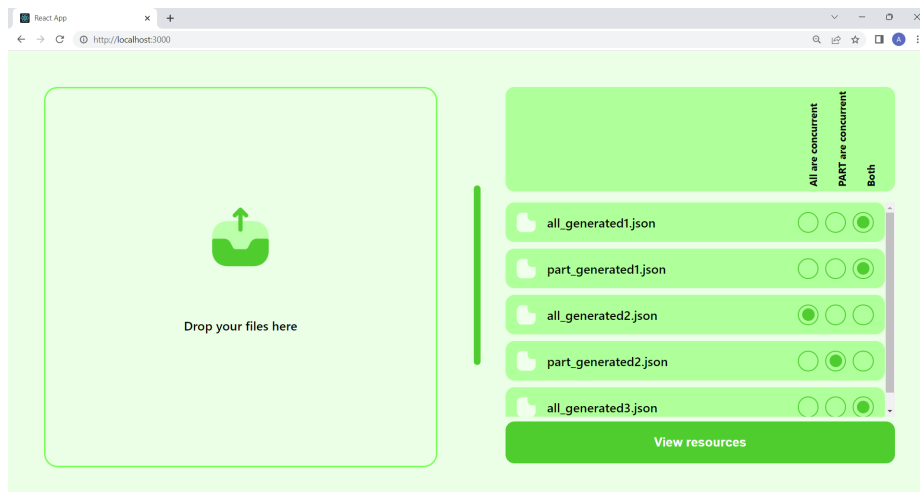


Figure 4: System’s main GUI.

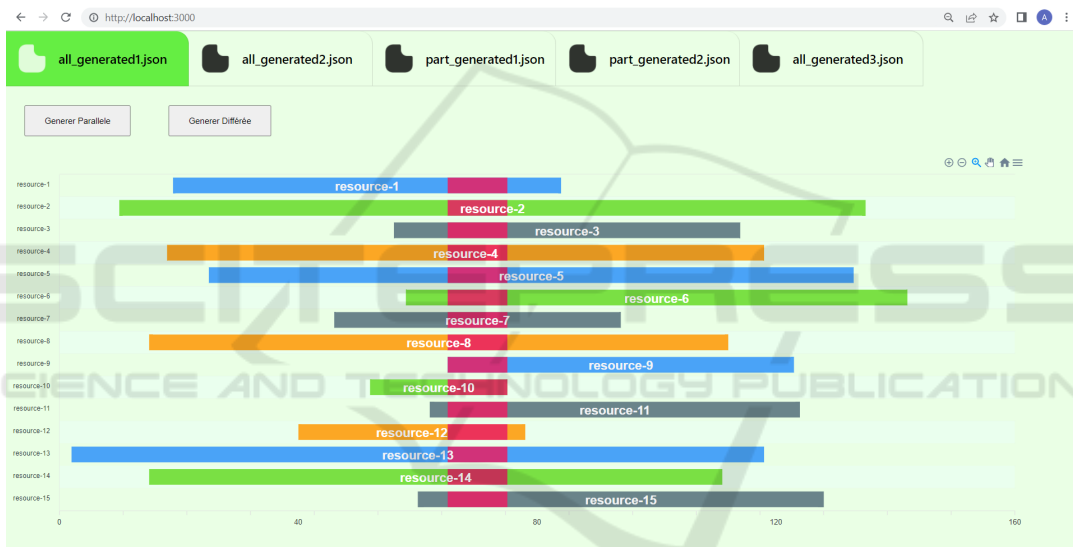


Figure 5: Representation of scenario 1’s resources’ availability-time intervals.

6.2 Tests and Evaluation

Due to the unavailability of datasets that would cater for our needs, we generated 2 separate datasets containing each 15, 150, 1500, 15000, and 150000 primitive resources along with their respective availability-time intervals. The scenarios that were tested and evaluated are as follows.

Scenario 1 (“All are concurrent”) targets a consumption flow that features concurrent consumption of all primitive resources (Fig. 5). The dataset of this scenario, i.e., UPR formatted as per Listing 1, is the result of Algorithm 2 whose inputs are a given time interval $([b, e])$ and a number of primitive resources (n). To define each primitive resource’s availability-time interval in the dataset, *generateInter-*

val function uses a *randomize* function twice where *genBegin* and *genEnd* are pseudo-random numbers with $0 \leq genBegin \leq b$, $e \leq genEnd \leq max$, and *max* is an arbitrary value like $(b + e)$.

Listing 1: Excerpt of scenario 1-related dataset in JSON.

```

1 [{"id":2,"name":"resource-3","beginTime":56,"
   endTime":114,"shareable":false},
2 ...
3 {"id":13,"name":"resource-14","beginTime":15,"
   endTime":111,"shareable":false}]

```

A graphical representation of Listing 1 is shown in Fig. 5 after setting the number of primitive resources to 15. Some Allen’s time relations in this representation include *during* between resource-1 and resource-2, and *overlaps* between resource-4 and resource-5.

Input: b : begin-time interval, e : end-time interval, n : number of primitive resources to generate

Output: \mathcal{UPR} : set of primitives resources

```

1 begin
2   Initialize  $\mathcal{UPR}$  to  $\phi$ ;
3   foreach  $i \in [1, n]$  do
4     generateInterval( $b, e, b +$ 
5        $e, \text{genBegin}, \text{genEnd}$ );
6      $\mathcal{UPR} = \mathcal{UPR} \cup$ 
7        $\text{createResource}(\text{genBegin}, \text{genEnd})$ ;
8   end
9   return  $\mathcal{UPR}$ 
10 end

```

Algorithm 2: Dataset generation for scenario 1.

After completing the identification and adjustment stages, [65,75] is the common availability-time interval (Fig. 5:red band) ensuring the concurrent consumption of all the primitive resources. Using the system, the designer can check the availability-time interval of each primitive resource, zoom in/out the displayed results, and save these results in formats like *csv* and *png*.

Scenario 2 ("Part are concurrent") targets a consumption flow that features both concurrent and sequential consumption of primitive resources (Fig. 6). This scenario whose dataset is formatted as per Listing 2 required adjusting Algorithm 1 in a way that *meets* and *precedes* time-relations were supported this time. To generate the second dataset, Algorithm 2 is used again after adjusting *randomize* function where a start time is generated first subject to satisfying $0 \leq \text{genBegin} < e$. Then, an end time that satisfies $\text{genBegin} < \text{genEnd} \leq \text{max}$ is generated based on this start time.

Listing 2: Excerpt of scenario 2-related dataset in JSON.

```

1 [{"id":0,"name":"resource-1","beginTime":25,"
2   endTime":93,"shareable":false},
3   ...
4   {"id":14,"name":"resource-15","beginTime":54,"
5     endTime":55,"shareable":false}]

```

A graphical representation of Listing 2 is given in Fig. 6 where 15 primitive resources are included again with a maximum number among these resources to consume concurrently and the rest sequentially. Some Allen's time relations include *meets* between resource-2 and resource 11, and *overlaps* between resource-12 and resource-9. After completing the identification and adjustment stages, Fig. 6:red band indicates the availability-time interval for consuming each resource in the final consumption flow. The designer can roll the mouse over the red band to check when the consumption of a primitive resource will effectively occur. All the consumption whether sequen-

tial or concurrent will fall into [28,100] interval. During this interval, several gaps like]39, 54[and]55, 62[can be noticed where no consumption will happen. Outside these gaps, other availability-time intervals like [28, 39] and [54, 55] will correspond to consumption.

Testing both scenarios was done in compliance with unit testing in test-driven development (Beck, 2003) where for each test an output result is compared to an expected output to verify the correctness of this result. In conjunction with this testing, we evaluated the creation time of a composite resource's final consumption flow (\mathcal{FCF}) per scenario (Table 4). This time increases as the number of primitive resources increases too with scenario 2 requiring more processing than scenario 1.

Table 4: Creation time of a composite resource's \mathcal{FCF} .

# of resources	\mathcal{FCF} creation time (ms)	
	Scenario 1	Scenario 2
15	4	6
150	6	750
1500	80	2522
15000	10040	25540

7 CONCLUSION

This paper presented an approach for coordinating the consumption of primitive resources subject to consumption properties and time availabilities. These properties included unlimited, limited, limited-but-extensible, shareable, and non-shareable impacting their assignments to user-related jobs. The assignments are also dependent on time periods dictating when a primitive resource is available for use. Multiple primitive resources means multiple separate time-availabilities that raise different coordination challenges. To tackle these challenges, Allen's time algebra was used establishing time relations between time availabilities like overlaps, meets, and during. The paper also presented a system demonstrating the technical doability of a time-constrained, event-driven coordination of composing primitive resources. In term of future work, we would like to examine the impact of failures on primitive resources' time availabilities. Could these availabilities be adjusted when a primitive resource has a limited consumption-property, for example? Another future work is to assess the impact of transactional properties like pivot, retrievable, and compensatable on primitive resources' availability times and hence, integration into composite resources.

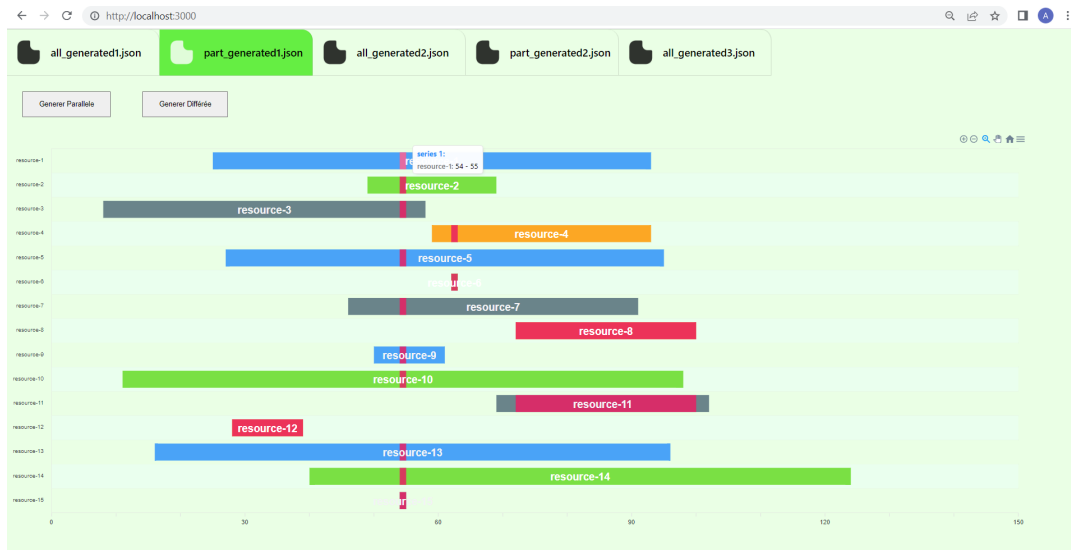


Figure 6: Representation of scenario 2's resources' availability-time intervals.

REFERENCES

- Akbarinia, R., Pacitti, E., and Valduriez, P. (2011). Best Position Algorithms for Efficient Top-k Query Processing. *Information Systems*, 36(6).
- Allen, J. (1983). Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11).
- Arias, M., Rojas, E., Munoz-Gama, J., and Sepúlveda, M. (2015). A Framework for Recommending Resource Allocation based on Process Mining. In *Proceedings of the 13th International Business Process Management Workshops held in conjunction with BPM'2015*, Innsbruck, Austria.
- Baker, T., Ugljanin, E., Faci, N., Sellami, M., Maamar, Z., and Kajan, E. (2018). Everything as a Resource: Foundations and Illustration through Internet-of-Things. *Computers in Industry*, 94.
- Beck, K. (2003). *Test-driven Development - by Example*. Addison-Wesley Signature Series. Addison-Wesley.
- de Weerd, M. and Clement, B. (2009). Introduction to Planning in Multiagent Systems. *Multiagent Grid Systems*, 5(4).
- Dimick, D. (2014). As World's Population Booms, Will its Resources be Enough for Us? <https://tinyurl.com/5an3dhez>.
- Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine.
- Hofman, W. (2015). Towards a Federated Infrastructure for the Global Data Pipeline. In *Proceedings of the 14th IFIP WG 6.11 on Open and Big Data Management and Innovation held in conjunction with I3E'2015*, Delft, The Netherlands.
- Janhunen, T. and Sioutis, M. (2019). Allen's Interval Algebra Makes the Difference. *CoRR*, abs/1909.01128.
- Langdon, C. S. (2003). The State of Web Services. *IEEE Computer*, 36(7).
- Lucchim, R., Millot, M., and Elfers, C. (2008). Resource oriented Architecture and REST: Assessment of Impact and Advantage on INSPIRE. JRC Scientific and Technical Reports EUR 23397 EN - 2008, JRC European Commission.
- Maamar, Z. and Al Khafajiy, M. (2021). Cloud-Edge Coupling to Mitigate Execution Failures. In *Proceedings of SAC'2021*, South Korea (online).
- Maamar, Z., Faci, N., Sakr, S., Boukhebouze, M., and Barnawi, A. (2016). Network-based Social coordination of Business Processes. *Information Systems*, 58.
- Maamar, Z., Sellami, M., and Masmoudi, F. (2021). A Transactional Approach to Enforce Resource Availabilities: Application to the Cloud. In *Proceedings of RCIS'2021*, Cyprus (online).
- Maamar, Z., Yahya, F., and Benammar, L. (2022). On the Use of Allen's Interval Algebra in the Coordination of Resource Consumption by Transactional Business Processes. In *Proceedings of ENASE'2022*, Portugal (online).
- Mell, P. and Grance, T. (2011). The NIST Definition of Cloud Computing. Technical Report 800-145, National Institute of Standards and Technology (NIST).
- Park, G. and Song, M. (2019). Prediction-based Resource Allocation using LSTM and Minimum Cost and Maximum Flow Algorithm. In *Proceedings of ICPM'2019*, Aachen, Germany.
- Schreiber, G. and Raimond, Y. (2014). RDF 1.1 Primer. Technical report, World Wide Web Consortium.
- Sindhgatta, R., Ghose, A. K., and Dam, H. K. (2016). Context-aware Analysis of Past Process Executions to Aid Resource Allocation Decisions. In *Proceedings of CAiSE'2016*, Ljubljana, Slovenia.

- Stefanini, A., Aloini, D., Benevento, E., Dulmin, R., and Mininno, V. (2020). A Data-driven Methodology for Supporting Resource Planning of Health Services. *Socio-Economic Planning Sciences*, 70.
- Weske, M. (2012). Business Process Management Architectures. In *Business Process Management*. Springer.
- Zhao, W., Liu, H., Dai, W., and Ma, J. (2016). An Entropy-based Clustering Ensemble Method to Support Resource Allocation in Business Process Management. *Knowledge and Information Systems*, 48(2).
- Zhao, W., Pu, S., and Jiang, D. (2020). A Human Resource Allocation Method for Business Processes using Team Faultlines. *Applied Intelligence*, 50(9).

