
Jinho On, Sujeong Woo, Moonkun Lee
Department of Computer Engineering
Chonbuk National University
Jeonju, Jeonbuk, Republic of Korea, 561-756
{jijinghott, wpig04}@gmail.com, moonkun@jbnu.ac.kr

Abstract—This paper presents a new approach for workflow design in cloud computing.

Generally the workflow design in cloud computing is specified by BPEL/BPMN, which is transformed into the existing formal methods for analysis and verification of the design. However the main paradigms of the existing methods reveal some limitations to the design due to their structural characteristics: process algebras mainly focus on in-the-large (ITL) views, and, reversely, state machines mainly focus on in-the-small (ITS) views. Therefore it is necessary to transform ITL views to ITS views, or vice versa, based on some equivalence relations. How can we avoid this notion of the equivalence and transformations to achieve some maximum integration views of the design?

A new visual formal method, called Onion, is presented in this paper to integrate these two different ITL and ITS views in a single view. In Onion, processes and their transitive actions are graphically represented in one single entity, just like those of a real onion. Further the temporal properties of actions in the processes are specified in a geo-temporal space. Once these are done, the requirements for the design are graphically specified on the processes and actions using a visual logic. Finally, the design is analyzed and verified through simulation in order to see whether it satisfies the requirements and restriction.

The comparative study shows that the Onion approach is very effective and efficient for BPEL/BPMN and overcome some limitations of BPMN/BPMN.


I. INTRODUCTION

This paper presents a new method for workflow design in cloud computing.

Generally the workflow design in cloud computing is specified BPEL[1] and BPMN[2], which is the visual representation of BPEL. Once the design is completed, it is analyzed and verified by the existing formal methods[3][4][5].

However the existing methods reveal some limitations to the cloud computing due to their structural characteristics, that is, of process algebras and state machines[6][7].

The cloud computing requires strong visualization in process modeling and workflow design, especially, that of process mobility, interactions, reconfiguration, requirements, verification, etc. in the target geo-temporal space[8].

However, the methods in process algebras seem to fail to show the detailed views of the designs since they focus mainly on in-the-large (ITL) views. Reversely, the methods in state machines seem to fail to show the abstract views of the designs since they mainly focus on in-the-small (ITS) views.

Due to these reasons, the equivalence between process algebra and state machine has been a critical issue to show both ITL and ITS views in one design.

The main objective of the research in this paper is this: How to overcome these limitations, satisfying all the requirements of the workflow design in cloud computing?

The method presented in this paper to achieve this objective is the integration of these two different representations in a new visual language, namely, Onion [9], with the basic properties of processes, timed actions (communications and movements), as well as hierarchical structure, in a geo-temporal space.

The main distinctive characteristic of Onion is the representation of processes, just as in process algebra, and their transitive actions, just as in state machine, in one singular entity. That is, each process in the design is represented as a process node in Onion, and the interactions and movements of the process are represented as the layered circular leaves of the node, just like those of a real onion, in a geographical space.

This representation shows both ITL and ITS views of the design, and satisfies the requirements of the design, that is, the visualization of process mobility, interactions, reconfiguration, etc.

Once it is done, the temporal properties of interactions and movements of the processes are specified on a geo-temporal space, which is the expansion of the geographical space to the temporal dimension. Each timed action is specified in a block with the temporal attributes, such as, ready time, timeout, execution time, deadline, etc.

Once the temporal properties are specified, the requirements of the processes and actions are graphically specified with a visual logic on geo-temporal blocks of processes and their actions.

Finally the static and dynamic verification of the requirements are performed and displayed on the geo-temporal space.
The overall approach in the paper is shown in Fig. 1:

- Firstly, an initial workflow design for a target service from cloud computing is specified in BPEL/BPMN and translated into processes and actions in Onion on a geographical space.
- Secondly, the extended features, i.e., mobility, interactions, and control of the processes in the design are added to the translated design on the geographical space.
- Thirdly, the temporal properties of the actions, i.e., communications and movements, are specified visually on the geo-temporal space.
- Fourthly, the requirements of the processes and their actions are specified visually on the geo-temporal space with a visual logic.
- Finally, the requirements are visually verified, both statically and dynamically, on the geo-temporal space.

All these processes are performed visually with the support of the Onion System [9].

The paper is organized as follows. Sections II and III present the basic theory of Onion. Section IV presents an iCloud example. In Section V, the approach will be comparatively analyzed with other approaches. Finally conclusions will be made.

II. ONION LANGUAGE

A. Onion Textual Language (OTL)

OTL is a universal process algebra in text for Onion Visual Language (OVL). The basic syntax and semantics of OTL are collected from CSP[10], CCS[11], π-calculus[12] and MA[13]. The extended features are the notions of mobility and process control.

The syntax of OTL is defined as follows:

The syntax of OTL is defined as follows:

\[ P ::= \alpha \cdot P^r \]
\[ \ | \ P \mid Q \ | P + Q \ | P \cdot Q \]
\[ P \cdot \langle n \rangle \ | P \cdot \langle b \rangle \]
\[ P \cdot [V] \ | P / [V] \]
\[ P \triangleright Q \]

\[ \alpha' ::= \alpha \ (b \rightarrow \alpha) \ ; \]
\[ P' ::= P \ (b \rightarrow P) \ ; \]
\[ \alpha ::= 0 \ | \ e \ | \sigma \ | \delta \ | \theta \ | r \ | \lambda \ ; \]
\[ \sigma ::= c! x \ | c ? x \ | c ! ? x \ | c ? ? x \ ; \]
\[ \delta ::= \text{in} Q \ | \text{out} Q \ | \text{get} Q \ | \text{put} Q \ ; \]
\[ \theta ::= \text{exits} \ | \text{exita} \ | \text{kills} Q \ | \text{killa} Q \ ; \]

Note that \( b, c \) and \( V \) are a Boolean condition, a channel name, and a list of variables to hide, respectively.

1. \( P \) : A process in a sequence of actions.
2. \( \alpha' \) : A conditional action, which is an action \( \alpha \) with \( b \).
   (i) \( 0 \) : No action.
   (ii) \( e \) : An empty action. If \(-b \), then \((b \rightarrow \alpha) = e\).
   (iii) \( \sigma \) : Communication actions.
      (a) \( c! x \ | c ? x \) : Asynchronous send/receive.
      (b) \( c ! ? x \ | c ? ? x \) : Synchronous send/receive.
   (iv) Movement actions (\( \delta \)):
      (a) \( \text{in} Q \ | \text{out} Q \) : Active or autonomous-in/out.
      (b) \( \text{get} Q \ | \text{put} Q \) : Passive or heteronomous-in/out.
   (v) Process termination actions (\( \theta \)):
      (a) \( \text{exits} \ | \text{exita} \) : self-termination.
      (b) \( \text{kills} Q \ | \text{killa} Q \) : Synch/asynch kill.
   (vi) Thread control actions:
      (a) \( P' \mid Q' \) : Parallel.
      (b) \( P' + Q' \) : Nondeterministic choice.
      (c) \( P' \cdot Q' \) : Sequential.
      (d) \( P' \cdot \langle n \rangle \) : Recursion. \( P' \cdot \langle n \rangle = P' \cdot (P' \cdot \langle n-1 \rangle) \), \( n \geq 1 \).
      \( P' \cdot \langle 1 \rangle = P' \), \( P' \cdot \langle 0 \rangle = 0 \).
   (vii) Other actions:
      (a) \( P \triangleright Q \) : Exception handling action:
      (b) \( P \cdot [V] \ | P / [V] \) : hide and reveal.

Note that the timing properties of an action is represented by \( \alpha_{r,0,e,d,p} \), where \( r, 0, e, d \) and \( p \) represent ready time, timeout, execution time, deadline, and, optionally, period, of the action, respectively.

B. Onion Visual Language (OVL)

OVL is a visual representation for OTL.

In OVL, a process is represented by a node, and its actions by a sequence of nested round layers, just like those of a real onion. For example, \( P ::= \alpha \cdot \beta \cdot \gamma \), that is, a process (\( P \)) consisting of three sequential actions (\( \alpha, \beta, \) and \( \gamma \)), is represented as shown in Fig. 2.
Note that that actions are denoted in two different types of circles: straight and dotted. These imply *active* ($\alpha$) and *inactive* ($\beta$, $\gamma$) actions. A process with an outermost active action is defined as an *active process*, and a process with the outermost inactive as an *inactive*. Note that all the conditional and nondeterministic actions are inactive.

For communication, a port is defined and represented by a dot on a circular action leaf. A communication between two processes is represented by a straight arrow between two ports. The synchronicity/asynchronicity of communication is distinguished by open/close mark on the dot. Note that only the arrow remains dotted unless both actions of two processes for a communication are active. Fig. 3 shows $\alpha := x!m$, $\beta := x?m$ and $\gamma := x?m$.

An *autonomous* movement is represented by a curved arrow with closed round/arrow marks at the ends, and a *heteronomous* movement by a curved arrow with open round/arrow marks. Once a movement occurs, the process will be relocated in another place, called *future process*, as the target of the movement. For example, Fig. 4 shows both autonomous and heteronomous movement actions. Note that the targets of the arrows are future processes.

A process can be nested in another process by visual containment.

### C. Transformation of BPEL Key Statements Onion

Table 1 shows the relations between BPEL key statements to Onion, extended from those to LOTOS in [14]. This allows the basic BPEL specification of workflow design to be translated to Onion. Once it is done, the further specification of mobility and reconfiguration of processes in the design is possible in Onion.

<table>
<thead>
<tr>
<th>Sample BPEL</th>
<th>Sample LOTOS</th>
<th>OTL</th>
<th>OVL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1&lt;br&gt;<code>&lt;...act1...&gt;</code>&lt;br&gt;<code>&lt;act1&gt;</code>&lt;br&gt;<code>&lt;assign&gt;</code>&lt;br&gt;<code>&lt;copy&gt;</code>&lt;br&gt;<code>&lt;from expression=&quot;5&quot;/&gt;</code>&lt;br&gt;<code>&lt;to var=&quot;x&quot;/&gt;</code>&lt;br&gt;<code>&lt;copy&gt;</code>&lt;br&gt;<code>&lt;assign&gt;</code>&lt;br&gt;<code>&lt;...act2...&gt;</code></td>
<td><code>act1...; exit(5)</code>&lt;br&gt;<code>accept x: Nat in ...act2...</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2&lt;br&gt;<code>&lt;receive ... variable=&quot;m&quot;/&gt;</code>&lt;br&gt;<code>&lt;receive&gt;</code>&lt;br&gt;<code>&lt;copy&gt;</code>&lt;br&gt;<code>&lt;from variable=&quot;x&quot;/&gt;</code>&lt;br&gt;<code>&lt;to var=&quot;m&quot;/&gt;</code>&lt;br&gt;<code>&lt;receive&gt;</code>&lt;br&gt;<code>&lt;reply&gt;</code>&lt;br&gt;<code>&lt;reply&gt;</code>&lt;br&gt;<code>&lt;sequence&gt;</code>&lt;br&gt;<code>&lt;...act1...&gt;</code>&lt;br&gt;<code>&lt;...act2...&gt;</code>&lt;br&gt;<code>&lt;sequence&gt;</code></td>
<td><code>g?m: Nat;</code>&lt;br&gt;<code>g?m</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3&lt;br&gt;<code>&lt;receive ... variable=&quot;m&quot;/&gt;</code>&lt;br&gt;<code>&lt;receive&gt;</code>&lt;br&gt;<code>&lt;copy&gt;</code>&lt;br&gt;<code>&lt;from variable=&quot;x&quot;/&gt;</code>&lt;br&gt;<code>&lt;to var=&quot;m&quot;/&gt;</code>&lt;br&gt;<code>&lt;receive&gt;</code>&lt;br&gt;<code>&lt;reply&gt;</code>&lt;br&gt;<code>&lt;reply&gt;</code>&lt;br&gt;<code>&lt;sequence&gt;</code>&lt;br&gt;<code>&lt;...act1...&gt;</code>&lt;br&gt;<code>&lt;...act2...&gt;</code>&lt;br&gt;<code>&lt;sequence&gt;</code></td>
<td><code>g?m: Nat;</code>&lt;br&gt;<code>g?m</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4&lt;br&gt;<code>&lt;while condition=&quot;bpws:getVariableData(x)\geq 0&quot;&gt;</code>&lt;br&gt;<code>&lt;...act1...&gt;</code>&lt;br&gt;<code>&lt;...act2...&gt;</code>&lt;br&gt;<code>&lt;while&gt;</code></td>
<td><code>P^x(x\geq 0);</code>&lt;br&gt;<code>P:=act1;</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
III. GEO-TEMPORAL SPACE AND VISUAL LOGIC

A. Onion Geo-Temporal Space (GTS)

The Geo-Temporal Space (GTS) for Onion is the representation of geographical space over temporal expansion in Onion. Geographical space (GS) is the one-dimensional vertical representation of OVL. Temporal Space (TS) is the one-dimensional horizontal representation of execution of processes and actions in OVL. Consequently, GTS is the two-dimensional representation of OVL.

A process $P$ in GS is represented by a vertical line between two geographical points, i.e., $t$ (at the top) and $b$ (at the bottom), and is denoted by $P(t,b)$. Any nested process should reside in the space of its parent process in the same pattern. Examples are shown in Fig. 5. By default, $P(t,b)$ is represented by $P$.

![Figure 5. GS for P.](image)

![Figure 6. TS over GS.](image)

The TS over GS for a process $P$ is defined as a geo-temporal block (GTB), namely, process GTS, and represented by a horizontal space over the GS. There are two types of GTB: discrete and continuous. The discrete space is GTB at a discrete time, $t_i$, i.e., $P_{[t_i]}$. The continuous space is GTB at the continuous time between $t_i$ and $t_j$ , i.e., $P_{[t_i,t_j]}$, as shown in Fig. 6.

A timed action of a process $P$ is also represented by a GTB namely, action GTS. Its timing requirements are based on the definition of the timed action: $r$, $t_0$, $e$, $d$ and $p$. The example of the actions for $P := \alpha \beta \gamma$ is shown in Fig. 7. The GTBs of the actions are represented with dotted lines since they are defined to be inactive. Note that the types of actions are denoted by the squared yellow marks in legend.

![Figure 7. Action with Timing Requirements.](image)

A timed interaction between $P$ and $Q$ is represented by a direct edge between action GTBs in process GTBs, as shown in Fig. 8.

![Figure 8. Timed Interaction (Synch Communication).](image)
B. Visual Logic (VL) on Geo-Temporal Space

Visual Logic (VL) is a language to specify visually the requirements of processes and actions in GTS. VL defines the requirements as some inclusion or precedence relations between/among process and action GTBs in GTS. The syntax and semantics of VL are described in Table II. Note that A and B in the table are process GTBs in geographical requirements, or action GTBs in temporal requirements.

<table>
<thead>
<tr>
<th>Geographical Requirements</th>
<th>Syntax</th>
<th>Semantics</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>A ⊆ B</td>
<td>A can be in B. B can be in A.</td>
<td>A ≠ B</td>
<td>A cannot be in B. B cannot be in A.</td>
<td></td>
</tr>
<tr>
<td>A ⊆ B</td>
<td>A can be in B. B cannot be in A.</td>
<td>A ⊘ B</td>
<td>B should be in A, always.</td>
<td></td>
</tr>
<tr>
<td>A ⊆ B</td>
<td>B can be in A. A cannot be in B.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Temporal Requirements</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>A ⊳ B</td>
<td>Action A should be performed before B in time.</td>
<td>Actions A and B can be performed concurrently in the same time period.</td>
</tr>
<tr>
<td>A ⊳ B</td>
<td>Action B should be performed after A.</td>
<td>Actions A and B cannot be performed concurrently in the same time period.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interval</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open interval</td>
<td></td>
<td>Close interval</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>−</td>
<td>Condition for requirements: time, frequency, behavior, etc.</td>
<td></td>
</tr>
</tbody>
</table>

Note that requirements can be expressed recursively in conjunctive and disjunctive forms.

![Figure 9. Example of VL on GTS](image9)

Fig. 9 shows some examples in VL on a GTS. For example, $R_1$ to $R_6$ define the following requirements:

- $R_1$: Process $A$ must reside in Process $B$ all the time.
- $R_2$: Action $b_1$ in Process $B$ must precede Action $c_1$ in Process $C$.
- $R_3$: Actions in $TS_2$ of $C$ and actions in $TS_3$ of $D$ can execute concurrently.
- $R_4$: The total execution time of actions in $TS_4$ of $D$ should be less than 100 time units.
- $R_5$: $C$ cannot move into $A$ within the $TS_1$ interval of $A$.
- $R_6$: All the actions of $D$ have to be terminated in 600 time units.

Note that requirements can be expressed recursively in conjunctive and disjunctive forms.

IV. AN EMS EXAMPLE

A. 911 Service in EMS

911 service is the main service in Emergency Medical Service (EMS), where, in case of a medically urgent situation occurred to Patient in House, EMS Center is informed and sends a message to 911 to transport the patient to Hospital, and, upon the message, an ambulance from the 911 go to the house and take the patient to the hospital in time.

![Figure 10. Workflow model example in BPMN](image10)

This service can be described in BPMN as shown in Fig. 10, where the movements of ambulances and patients are represented in the form of message-passing due to the limitation of expressivity for movements in BPMN. Consequently there are considerable difficulties in specifying the movements of patients, ambulances and doctors in the EMS example. Further there is no way to display the overall reconfigurations of the example in timely progression.

B. OTL and OVL

In the Onion method, a BPMN specification for the EMS can be translated into OTL, and the movements and actions of the processes in the EMS are specified with related data further in OTL and OVL as shown in Fig. 11 and 12.

![Figure 11. OVL for iCould Example](image11)
In OVL, the key players of the EMS can be visually configured as shown in Fig. 11, and their interactions and movements are visually recognizable.

The pictorial description is as follows:

- \(a_1\): A critical event is detected from a patient’s sensor and sent to the EMS center. The center informs both a 911 center and a hospital, where the patient has been under treatment, of the patient’s situation.
- \(a_2\): An ambulance from the 911 goes to the patient’s house.
- \(a_3\): The patient is carried out of the house and gotten on the ambulance.
- \(a_4\): The ambulance goes to the hospital.
- \(a_5\): The patient is gotten off the ambulance and carried into the emergency room of the hospital.
- \(a_6\): A medical doctor in the hospital is informed of the patient with a message.
- \(a_7\): The doctor goes to the emergency room.

Note that, in OVL, the movements, communications, and reconfiguration of processes can be represented in hierarchically organized structure.

As stated, the most important feature of OVL is the integrated representation of ITL and ITS views of the service.

C. Onion GTS

Once the EMS is specified in OVL, it is necessary to define temporal properties of actions of processes in the EMS. Fig. 13 shows GTBs of each processes and actions in them, as well as the interactions, from \(a_1\) to \(a_7\), among them. Note that the temporal properties of each action are scaled over its temporal dimension. Its temporal scale is presented at the top of the figure.

D. VL

The requirements for the example are specified visually in Fig. 14. These imply the following conditions to be satisfied:

- \(R_1\): The doctor has to arrive at the emergency room before the patient is carried into the room.
- \(R_2\): The patient has to be carried to the hospital in the time units of 400 after he is gotten on the ambulance.

E. Analysis and Verification

In the Onion method, the above requirements can be verified, dynamically. The dynamic verification implies the verification based on run-time analysis. The results of the verifications can be visually displayed on the verified GTS.

For example, Fig. 15 shows the results of the dynamic verification for the example. The similar results are generated. The difference is that it shows the run-time results as it is being executed. The figure shows the results of the execution at the time units of 700, as shown at the top of the figure. All the executed GTBs are represented as the straight lines.
V. COMPARISON WITH OTHER METHODS

Generally workflow design in cloud computing with BPEL/BPMN consists of the following steps: 1) specification, 2) analysis, and 3) verification. Once the design is specified in BPMN, the specified design is transformed to some formal methods for analysis and verification. The properties of the design to be analyzed and verified are basically dependent on the capability and characteristics of the formal methods. As stated, process algebras mainly focus on ITL views, and, conversely, state machines mainly focus on in-the-small (ITS) views. Due to these characteristics, it is necessary to transform process algebras to state machines, or vice versa, to see both ITL and ITS views by means of some equivalence relations.

Onion attempted to handle this problem by integrating both views in one view in Onion. Further Onion tried to visualize all the contents of specification, requirements, analysis and verification of the design, as well as all the temporal properties of the contents, using OVL, GS, GTS, GTB, Visual Logic, etc.

VI. CONCLUSIONS AND FUTURE RESEARCH

This paper presented the Onion method for workflow design in cloud computing. The method has the following innovative features:

- Visual specification of processes and their actions/interactions on a geographical space,
- Notion of GTBs for processes and their actions on GTS, as well as interactions among processes,
- Notion of visual logic and graphical requirements on GTBs in GTS, and
- Notion of visual analysis and verification.

The paper also demonstrated that the method would be well suited for the design by translating BPMN into Onion and going through analysis and verification. It also showed that some limitations of BPEL/BPMN could be overcome by using Onion.

The future research includes the development of the complete set of the Onion tools, its application to the real industrial examples, etc.

ACKNOWLEDGMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2010-0023787), and by research funds of Chonbuk National University (2011) and by second stage of Brain Korea 21 Project in 2012.

REFERENCES