There are three types of business processes:

1. Management processes, the processes that govern the operation of a system. Typical management processes include ”corporate governance” and ”strategic management”.

2. Operational processes, processes that constitute the core business and create the primary value stream. For example, taking orders from customers, and opening an account in a bank branch.

3. Supporting processes, which support the core processes. Examples include accounting, recruitment, call center, technical support.

A business process begins with a mission objective and ends with achievement of the business objective. Process-oriented organizations break down the barriers of structural departments and try to avoid functional silos.
A business process can be decomposed into several sub-processes,[1] which have their own attributes, but also contribute to achieving the goal of the super-process. The analysis of business processes typically includes the mapping of processes and sub-processes down to activity level.

Business Processes are designed to add value for the customer and should not include unnecessary activities. The outcome of a well-designed business process is increased effectiveness (value for the customer) and increased efficiency (less costs for the company).

Business Processes can be modeled through a large number of methods and techniques. For instance, the Business Process Modeling Notation is a Business Process Modeling technique that can be used for drawing business processes in a workflow.
Web Services Business Process Execution Language, BPEL is an XML-based language for specifying business process behaviour in a service-oriented architecture (SOA), based on interactions between a group of Web services partners.

Programmers use BPEL to define how a business process that involves web services will be executed. BPEL messages are typically used to invoke remote services, orchestrate process execution and manage events and exceptions.

BPEL is often associated with Business Process Management Notation (BPMN), a standard for representing business processes graphically. In many organizations, analysts use BPMN to visualize business processes and developers transform the visualizations to BPEL for execution.

BPEL was standardized by OASIS in 2004 after collaborative efforts to create the language by Microsoft, IBM and other companies.
BPEL is not a formal language. In order to analyse business processes with formal methods, BPEL has been translated to various formal languages.


Also to Lotos NT (Thivolle: Langages modernes pour la modélisation et la vérification des systèmes asynchrones. PhD thesis, Grenoble University 2011). Lotos NT is a simplified version of E-Lotos and E-Lotos is an extension of Lotos with time and new data type formalism.

These works have two major drawbacks. First, they provided only a partial coverage of BPEL. Secondly, they do not take time into account.

The lack of temporal aspects in formalization reduces the accuracy of the model and does not allow the analysis of quantitative properties.

Only few approaches have been developed to formalize the timed aspects induced by the interaction of Web services. Article Fares, Bodeveix, Filali: Verification of Timed BPEL 2.0 Models, Enterprise, Business-Process and Information Systems Modeling 81, 261-275 (2011), used FIACRE specifications to model timed aspects of BPEL.

Fiacre stands for ”Format Intermédiaire pour les Architectures de Composants Répartis Embarqués”, french for ”Intermediate Format for the Embedded Distributed Component Architectures”. Fiacre is a formally defined language for representing compositionally both the behavioural and timing aspects of embedded and distributed systems for formal verification and simulation purposes.
The fiacre language was designed in the framework of several projects gathering industry and academics partners, by a consortium of scientists from the OLC then VerTICS team of LAAS-CNRS, the VASY project of INRIA and the ACADIE team of IRIT.


We will follow the paper Chama, Belala and Saïdouni: FMEBP: A Formal Modeling Environment of Business Process, ICIST 2014. The paper introduces D-Lotos and an environment which translates BPEL specifications to D-Lotos.
D-Lotos is a high-level specification language modelling real-time systems. It extends basic Lotos with timing and urgency constraints giving to action a duration. The grammar of D-Lotos is as follows:

1. stop
2. exit\{d\}
3. $\Delta^d E$
4. $X[L]$
5. \texttt{g@t[SP]; E}
6. \texttt{i@t \{d\}; E}
7. $E[ ]E$
8. $E | [L] | E$
9. hide $L$ in $E$
10. $E >> E$
11. $E | [L>] | E$
12. $E[> E$
Now we try to explain the semantics. Let $a$ be an action, $E$ a behaviour expression and $d \in D$ a value in a countable time domain (for example, positive rational numbers).

- $a\{d\}$ means that the action $a$ must start its execution in the temporal domain $[0, d]$.
- $\Delta^d E$ means that no evolution of $E$ is allowed before a time delay equal to $d$.
- In $g@t[SP]; E$ (respectively $i@t \{d\}; E$), $t$ stores the time passed since the enabling of the action $g$ (respectively $i$) and which will be substituted by zero when this action finishes its execution.

Of the rest of the operators, only $E|L>|E$ demands an explanation. It means that $E$ evolves independently of $F$. Two cases may occur. First, $E$ terminates its execution and then the system terminates. Secondly, $E$ synchronizes with $F$ on an action in $L$. If this occurs, then $E$ is preempted and $F$ takes the control.
Web Services Business Process Execution Language (BPEL for short) is an XML based language for specifying business process behavior based on interactions between a group of Web services partners. This interaction occurs through <partnerLinks>. In BPEL, we have two parts of modeling.

The first is the abstract part. It is written in a WSDL document that describes the various Web services, their data types, port types and the messages exchanged between different partners without specifying any internal behavior.

Specifically, in BPEL each <partnerLinks> may contain two roles (<myRole> and <partnerRole>) typed by <portType>. In each <portType>, we can declare several operations that are used to receive (input) or to send (output) <messages>.

The second is the dynamic part which is described by two categories of activities:
Basic activities: are those which do not contain any other activity like <Receive>, <Reply>, <Invoke>, etc.

Structured activities: are obtained by composing other primitive and/or structured activities recursively. They are used to define the order in which their enclosed activities are executed. For example, sequential and parallel processing may be modeled respectively by <Sequence> and <Flow>. Finally, the <Scope> activity which allows the decomposition of the BPEL process into sub-processes.

A scope can be associated with a set of:

(a) Event handlers: which can be run concurrently and invoked when the corresponding event occurs (inbound messages events: <onEvent>, or alarm events: <onAlarm>);

(b) Fault handlers: to handle error messages when a fault occurred;
(c) *Compensation handler*: to compensate the scopes which are completed successfully in case where exceptions occur or a partner requests a backtrack;

(d) *Termination handler*: that controls the forced termination of a scope.
FMEBP is a formal environment that automates the transformation process of BPEL descriptions to D-LOTOS specifications. This platform is implemented with Java as a plugin to Eclipse development environment.

The input is a BPEL executable file, that is, an XML document which specifies the behavior of business process, while the output is a D-LOTOS process that formalizes the input BPEL file.

The user only needs to give a BPEL file as input, and then the plugin will automatically transform it into D-LOTOS process.

The obtained D-LOTOS process can be directly and automatically transformed into other low-level timed structures (such as Durational Action Timed Automata (DATA)), that can supply formal verification methods like model checking or bisimulation.
The user can then use UPPAAL to simulate the running process and check properties assessing the well system behavior.

The UPPAAL tool may be used for simulating the running of the process, as well as checking both qualitative and quantitative properties stated in temporal logic like TCTL (Timed Computational Tree Logic).

These properties concern: safety (something bad never occurs), liveness (something good eventually occurs) and deadlock freeness. By taking into account the duration of the actions, we can easily verify the behavior properties besides the time performance as well.
This part introduces the representation of basic BPEL activities in the specification language D-LOTOS. Basic activities are those which do not contain any other activity (for example, `<invoke>`, `<receive>`, and `<reply>` activity). In our approach, each BPEL simple activity is represented by a specific D-LOTOS process.

Consider the following mapping illustrated, where act denotes a basic BPEL activity, and g and d represent, respectively, guard and duration.

\[
\text{process BPELBasicActivity [act[(act)],...] := act\{g\}; exit Endproc}
\]

\(\text{act}\{g\}\) means that the action act has to begin its execution within \(g\) time units, and \(\text{act}[\tau(\text{act})]\) means that the necessary time for the complete execution of the action act is \(d = \tau(\text{act})\).
Structured activities are obtained by composing other primitive and/or structured activities recursively. They are used to define the order in which their enclosed activities are executed.

In Table 1, we define D-LOTOS processes that represent structured BPEL activities.

First of all, the <Sequence> and the <flow> activities in BPEL are formalized by the D-LOTOS sequential composition construct " and the two parallel composition operators \[ \| \ldots \| \]" or " || " respectively.

Intuitively, the D-LOTOS recursive process calls correspond either to the <while> activity enabling repeated execution as long as its boolean condition evaluates to “true” at the beginning of each iteration, or to the <repeatUntil> activity that is used to express repetitive execution.
In contrast to the <while> activity, the encapsulated activities of the <repeatUntil> activity are executed at least once as its boolean condition is evaluated at the end of each iteration.

Note that the provided condition in both of them (a boolean expression) is not represented in this formalization. Thus, it uses directly the non-deterministic choices between a recursive call (recursive behavior) and the exit action (ending behavior).

The branching conditions in the <if> activity might comprise of boolean expressions that are not represented in this formalization. Again, it uses directly the nondeterministic choices \([\]" (possibly multiple) to choose a possible activity which may be executed
Mapping of the Structured Activities III

- **Sequence**
  
  \[ \text{activity1} \ldots \rightarrow \text{activity2} \ldots \rightarrow \ldots \rightarrow \text{activityN-1} \ldots \rightarrow \text{activityN} \ldots \]

- **If**
  
  \[ \text{condition1;} \text{activity1} \ldots [\ldots] [\ldots] [\ldots] \]
  
  \[ \text{condition2;} \text{activity2} \ldots [\ldots] \text{exit} \]

- **While**
  
  \[ \text{process while} \ldots = \]
  
  \[ (\text{activity} \ldots \rightarrow \text{while} \ldots) [\ldots] \text{exit} \]

- **RepeatUntil**
  
  \[ \text{process repeatUntil} \ldots \]
  
  \[ \text{activity} \ldots \rightarrow (\text{repeatUntil} \ldots)[\ldots] \text{exit} \]

- **Flow and Links**
process flow [...] 
    activity1 [...] |[link]| activity2 [...] 
    |[link]| ... |[link]| activityn [...] 

- FaultHandlers

process faultHandlers [...] 
    catch1 [faultName1, ...]
    [] ...
    [] catchn [faultNameN, ...]
    [] catchAll [faults, ...]